119
94

Design of a PC based Data Acquisition System for a Switched Reluctance Motor

by

G. Chandramouli

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Krishnan Ramu
Dr. Krishnan Ramu, Chairman

Charles E. Nunnally
Dr. Charles. E. Nunnally

Jaime De La Ree
Dr. Jaime De La Ree

December, 1988

Blacksburg, Virginia

Design of a PC based Data Acquisition System for a Switched Reluctance Motor

by

G. Chandramouli

Dr. Krishnan Ramu, Chairman

Electrical Engineering

(ABSTRACT)

The Switched Reluctance Motors(SRM) have gained considerable attention in the variable speed drive market mainly due to the simple construction of the motor and the possibility of developing low cost converters and controllers. As these machines are under development, a considerable amount of research effort is directed to the experimental performance evaluation of the SRM drives. System efficiency, electromagnetic torque, torque ripple, output and losses are some of the required measurements.

The characterization of the SRM under wide operating conditions requires considerable data acquisition and processing. This thesis contains the design and development of a Personal Computer(PC) based Data Acquisition System(DAS) for a SRM drive. The design of measurement algorithms to evaluate the performance during steady state operation and the subsequent development of software based on these algorithms are presented. The DAS software is integrated with the existing controller software for research and prototype development. Experimental results from a functional laboratory prototype SRM drive are presented and discussed.

Acknowledgements

First of all, I would like to express my sincere thanks to Dr. Krishnan Ramu for being my advisor and guiding my thesis. I would also like to thank Dr.C.E. Nunnaly and Dr.J. De La Ree for agreeing to be in my committee.

Mr. Mang Xuesi helped me a lot during the development of hardware and software, and I deeply express my gratitude for him. I also appreciate the contributions made by Mr. Peter Materu and Mr. Aravind S. Bharadwaj at various stages of my thesis. Last but not least, I would like to thank my family and friends for their continued encouragement and support during the course of my graduate studies.

Table of Contents

List of Illustrations

1.0 Introduction

1.1 Problem statement and Solution method

Since the SRM has gained widespread attention in the variable speed drive market, it has become necessary to develop a package for the experimental performance evaluation of the SRM drive. Such an evaluation can be done by capturing the drive parameters by means of a DAS and post-processing the data using a performance evaluation software. But a general purpose DAS is not suitable for an SRM drive for the following reason. The parameters needed to evaluate the performance of the SRM drive are of varied nature. For example, the electromagnetic torque developed by the machine is dependent upon the rotor position as well as the phase current. The rotor position is obtained from a 10 bit position encoder while the phase current is obtained from a current transducer which is an analog input. Moreover, the time period over which the measurements are made must also be recorded from a timer. Presently, there is no DAS which is available to record the data for the performance evaluation of the SRM drive.

Hence, a PC based DAS which is integrated with the existing PC based SRM controller is proposed in this thesis. A PC compatible interface board which satisfies the requirements of both the controller and the DAS has been built and tested. Since the controller already runs on the PC and needs an interface board, there is no additional cost involved in building this DAS, thus saving an additional PC and a data acquisition board.

The following procedure is adopted in the proposed method:

1. Start the motor and bring it to the desired operating condition using the controller.
2. Give the command to run the DAS. Program control is automatically transferred to the DAS. The required number of data points are recorded and control is given back to the controller.
3. Stop the motor. The recorded data is stored on a disk.
4. Use the stored data to run the performance evaluation software.

Since timing is very critical and the controller has to run continuously to keep the motor running, on-line processing of the recorded data is not attempted. On-line processing can be done if the PC used for the implementation runs with a high speed clock. Some of the processed parameters like torque can be used by the controller for improving the performance of the system. However, off-line processing of the data is found adequate for the test set up.

1.2 Literature Survey

A number of papers have been published on different aspects of the SRM drive, describing the motor, converter and the controller. A comprehensive study of the SRM appeared as early as 1980 [1]. The performance of this drive is also compared with that of an induction motor drive. A simple, step-by-step design procedure similar to that adopted in conventional machines is presented earlier [2]. A good comparison between the performance of two different motors, a 6 stator pole 4 rotor pole (6/4) machine and a 12 stator pole 10 rotor pole (12/10) machine is presented in [3]. Differences in flux linkages, terminal inductance and the torque/ampere are discussed. A Computer Aided Design (CAD) approach, and its verification using finite element analysis technique has been implemented for the motor design [4].

Many different converter configurations have been proposed for the SRM drive [5,6]. The converters can have one switch per phase, two switches per phase, split dc supply or bifilar windings. The onset of saturation in the motor significantly affects the design and the rating of the converter [7]. The increase in converter kVA due to saturation is seen as a decrease in the input power factor. The steady state analysis of the motor converter combination is discussed in [8].

The control requirements of the SRM drive are different from that of synchronous and induction motors. A single chip microcontroller based control for a (8/6) SRM is discussed in [10]. The control has been implemented with speed and torque feedback loop. An Intel 8088 based control scheme has also been developed [11]. A personal computer based control is more attractive than the above schemes due to the avail-

ability of an user friendly interface through keyboard and monitor [12,13]. All the above mentioned control schemes make use of an absolute position encoder to detect the rotor position. This becomes very expensive for low power applications. Hence, it is preferable to implement sensorless control [14,15]. The dc link current is sensed and the pulse width and the turn-off angle are calculated on the basis of a predetermined profile.

Although many papers have addressed the issues of the SRM design, its converter and controller requirements, there have not been any publications on the determination of the performance of the motor by means of a data acquisition system. Such a system would make it possible to determine the motor characteristics under various operating conditions. This thesis attempts to develop such a system for the performance evaluation of the SRM under steady state operating conditions.

1.3 Organization of the thesis

The main objective of this thesis is to develop a PC based DAS integrated with the SRM controller to evaluate the performance of the SRM drive under steady state operating conditions. The organization of the chapters is discussed in the following paragraphs:

Chapter 2 discusses the SRM drive system. The construction and principle of operation of a prototype SRM is presented. Two different converter topologies that have been tested with the prototype motor are discussed. The basic ideas behind the im-

plementation of the controller are discussed. The concept of a PC based controller is introduced.

Chapter 3 explains the PC based controller in detail. The design of a PC compatible interface board catering to the needs of both the controller and DAS is discussed in detail. The features of the PC based controller are also discussed.

Chapter 4 discusses the implementation of the DAS. Since the integration of the DAS with the controller is of utmost importance, this chapter begins with this section. The motor parameters that are needed for performance evaluation and the method of acquisition of these parameters by the DAS is considered. Finally, resolving of timing conflicts between the DAS and the controller is discussed.

Chapter 5 discusses the off-line processing of the data acquired and stored by the DAS. The algorithms used for the evaluation of motor parameters like electromagnetic torque, output, efficiency, etc., and their subsequent implementation in software is discussed. The inductances for different positions and currents have been measured from the prototype SRM.

Chapter 6 discusses the results obtained by the DAS for different steady state operating conditions. The results are also compared with the dynamometer test results. Conclusions and recommendations for future study are discussed. The limitations of the existing system and the improvements that can be incorporated into it are also discussed.

2.0 The Switched Reluctance Motor Drive System

This chapter contains a discussion of the SRM and its principle of operation[5]. Two different converter topologies and the controller configuration are discussed. Further, the advantages of using a personal computer as a controller are discussed.

2.1 The SRM

The origin of the SRM can be traced back to 1842. However, it requires a controller and a power converter for its operation. Due to the lack of proper technology, it was not viable to build an SRM drive. But in the recent past, it has been possible to build relatively inexpensive converters and controllers. Hence, the SRM has drawn considerable attention in the variable speed drive market.

A 6 stator pole, 4 rotor pole (6/4) SRM is illustrated in Figure 1 on page 7. Both the rotor and the stator of the SRM have salient poles and hence referred to as a double

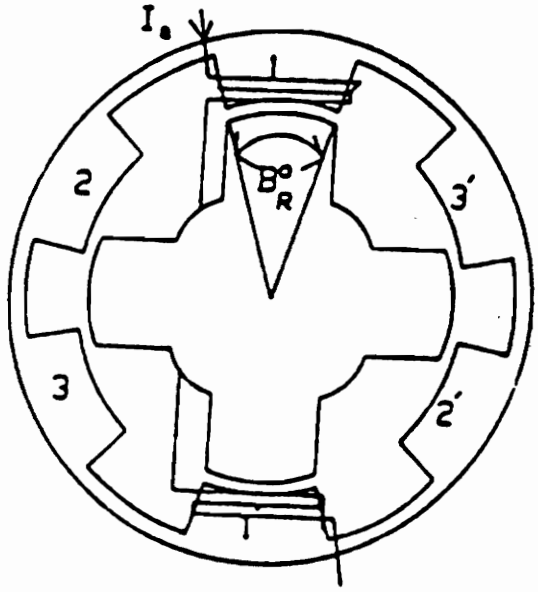


Figure 1. The Switched Reluctance Motor

salient machine. The stator poles carry windings while the rotor poles do not carry any windings. Whenever the diametrically opposite stator poles are excited, the nearest pair of rotor poles come into alignment with the stator poles. This is because, in a magnetic circuit, the rotor rotates to the minimum reluctance position when excited. At this instant, the other set of rotor poles are in the fully unaligned position. If the excitation is cut-off to the first pair of stator poles and the adjacent pair is excited, the other pair of rotor poles will move into complete alignment. Thus, by switching the currents into the stator windings in the proper sequence, a continuous motion of the rotor is obtained.

The production of torque in an SRM can be explained by referring to Figure 2 on page 9. The mechanical workdone in moving the rotor from $\theta = \theta_1$ to $\theta = \theta_2$ is given by the area enclosed between the two curves i.e.,

$$\delta\omega_m = \text{area } OABO \tag{2.1}$$

If T_e is the electromagnetic torque and $\delta\theta$ is the incremental angle, then

$$T_e = \frac{\delta\omega_m}{\delta\theta} \tag{2.2}$$

If the inductances are linear, it can be proved that

$$\delta\omega_m = \{L(\theta_2, i) - L(\theta_1, i)\}idi \tag{2.3}$$

Hence the incremental torque produced in moving the rotor from θ_1 to θ_2 is

$$\begin{aligned} \delta T_e &= \frac{\{L(\theta_2, i) - L(\theta_1, i)\}idi}{\theta_2 - \theta_1} \\ &= \{\frac{dL}{d\theta}(\theta, i)\}idi \end{aligned} \tag{2.4}$$

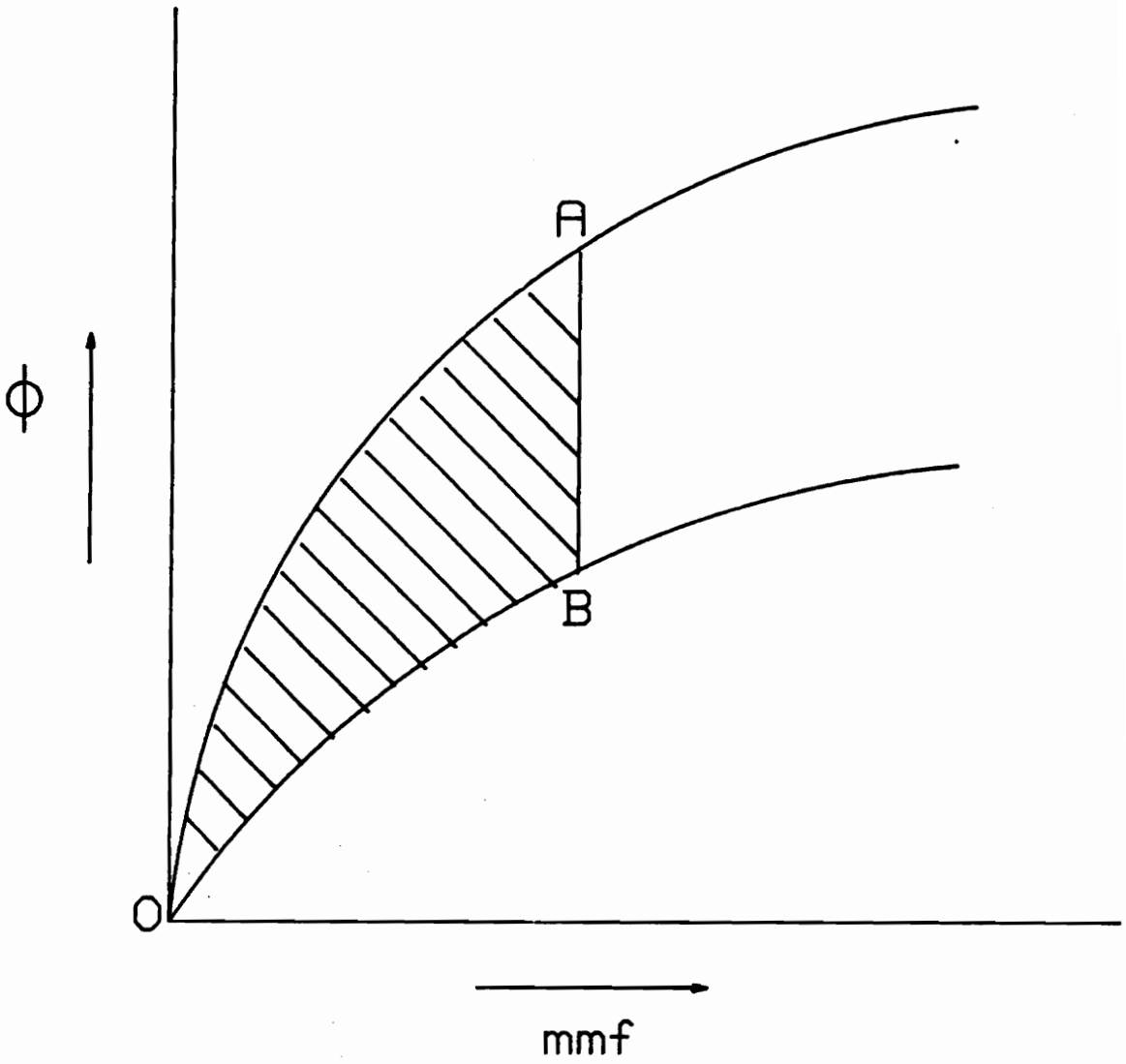


Figure 2. Flux vs mmf for various positions

Integrating equation (2.4) with the assumption that the inductance slope is a constant for current i,

$$T_e = \frac{dL}{d\theta}(\theta, i)\frac{i^2}{2} \tag{2.5}$$

From the above discussions the following conclusions can be drawn:

1. Since the rotor does not carry any windings, the cost of manufacturing the motor is reduced.

2. Since the torque is proportional to the square of the current, the current can be unidirectional to produce unidirectional torque. This has the advantage that only one power switch is needed per phase winding.

3. Due to the need for a controller and a converter for its operation, the SRM is inherently a variable speed drive.

2.2 Converter topologies

Since the torque in an SRM is independent of the current polarity, the SRM drives need only one power switch per phase. This is an advantage over the ac motor drives which require at least two switches per phase for current control. One such converter topology is shown in Figure 3 on page 11. Phase A is excited when T_m and T_1 are turned on. If the current exceeds the maximum limit, either T_1 alone or both

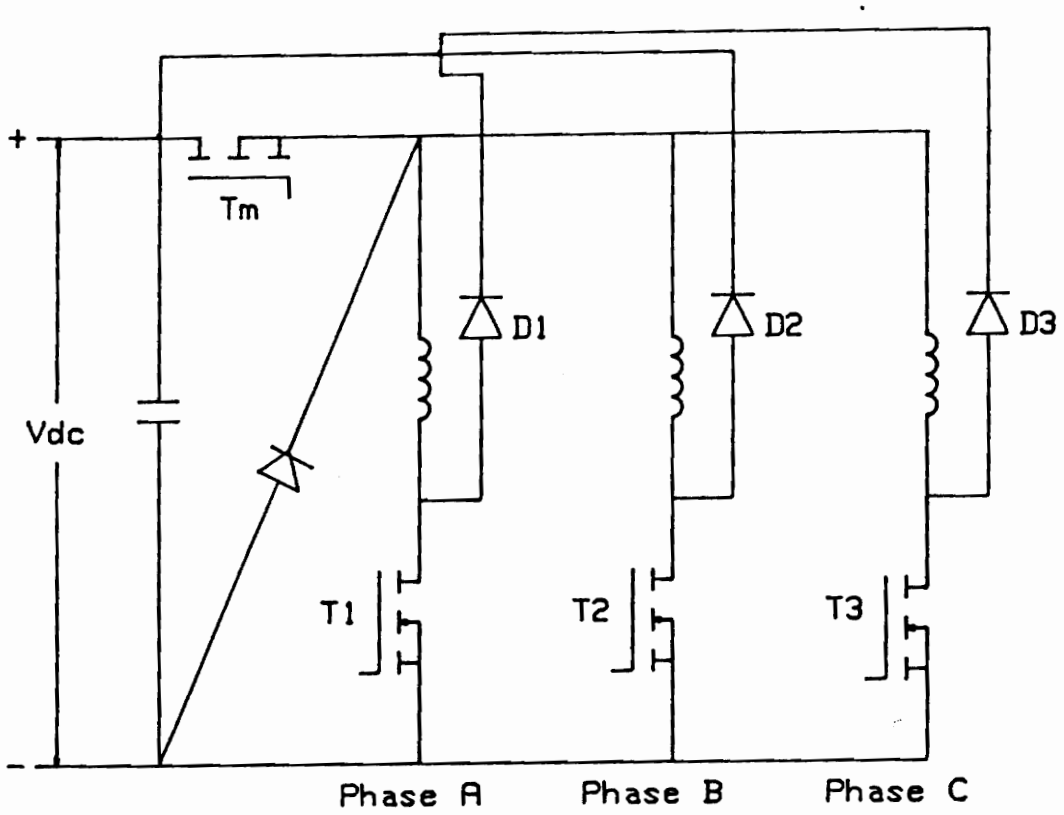


Figure 3. Converter for the SRM with one switch per phase

T_m and T_1 can be turned off. When only T_1 is turned off, the current freewheels within the winding, consequently taking a longer time to reduce to zero. This can create the possibility of developing negative torque. To overcome this problem, both T_m and T_1 are turned off. In this case, the energy stored in the winding is transferred back to the source, thus reducing the current rapidly to zero.

One problem associated with the previous converter topology is that, if phase A current does not reduce to zero before phase B is turned on, the current conduction time is prolonged which might result in the generation of negative torque. Another topology using two switches per phase overcomes this problem and is shown in Figure 4 on page 13. Phase A conducts when T_1 and T_2 are turned on. If the current exceeds the preset limits, both the switches are turned off resulting in regeneration. The current flowing in this phase is independent of the switching of the next phase. The disadvantage of this topology is that it uses more power switches than the one shown in Figure 3 on page 11. All future discussions and experimental verifications are done with reference to this topology.

2.3 SRM Controller

The control of the SRM is discussed with reference to Figure 5 on page 15 which shows the idealized inductance profile, current and voltage waveforms in one of the phase windings. For the prototype SRM, the rotor pole arc is 36° and the stator pole arc is 24°, giving a constant inductance profile of 12° between θ_3 and θ_4 when the stator pole is completely under the rotor pole. A constant slope is obtained for 24°

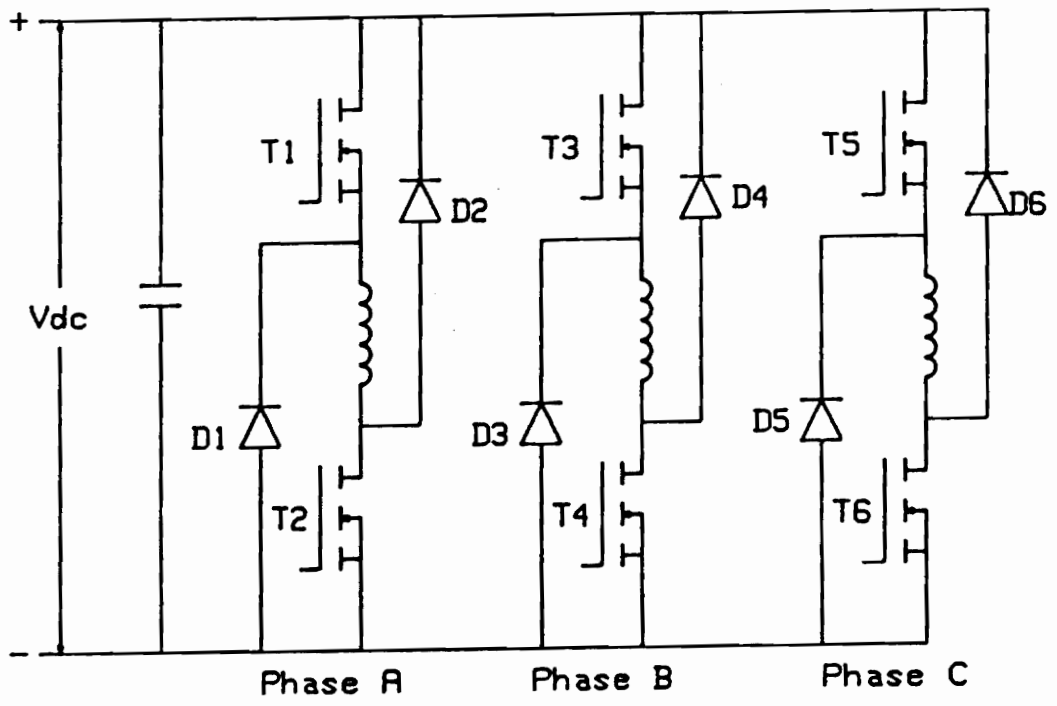


Figure 4. Converter for the SRM with two switches per phase

between θ_2 and θ_3, when the rotor pole begins to move under the stator pole. $(\theta_2 - \theta_1)$ is 15° in the fully unaligned position. One period of the inductance profile, θ_5 is given as

$$\theta_5 = \frac{360°}{N_r} \tag{2.6}$$

where N_r is the number of rotor poles.

From equation (2.5), it can be seen that the torque is proportional to the square of the current and $\frac{dL}{d\theta}$. In order to generate a constant torque, the current has to be maintained at the commanded level. This is done by means of hysteresis current control where the current is maintained within a small window around the commanded current. Maximum torque can be obtained from the machine if the current reaches the commanded value at θ_2. This is done by turning on the phase switches in advance to the rising inductance profile. This advance angle, called the rise angle, is given by

$$\theta_a = \frac{IL_{min}\omega_r}{V_d} \tag{2.7}$$

where,

I is the commanded current, A

L_{min} is the unaligned inductance, H

ω_r is the rotor speed, rad/sec

V_d is the dc bus voltage, V.

Since the torque is proportional to $\frac{dL}{d\theta}$, the current in a phase winding has to be reduced to zero before the beginning of the negative inductance profile. Hence the phase switches are turned off in advance of the falling inductance profile at θ_4. This

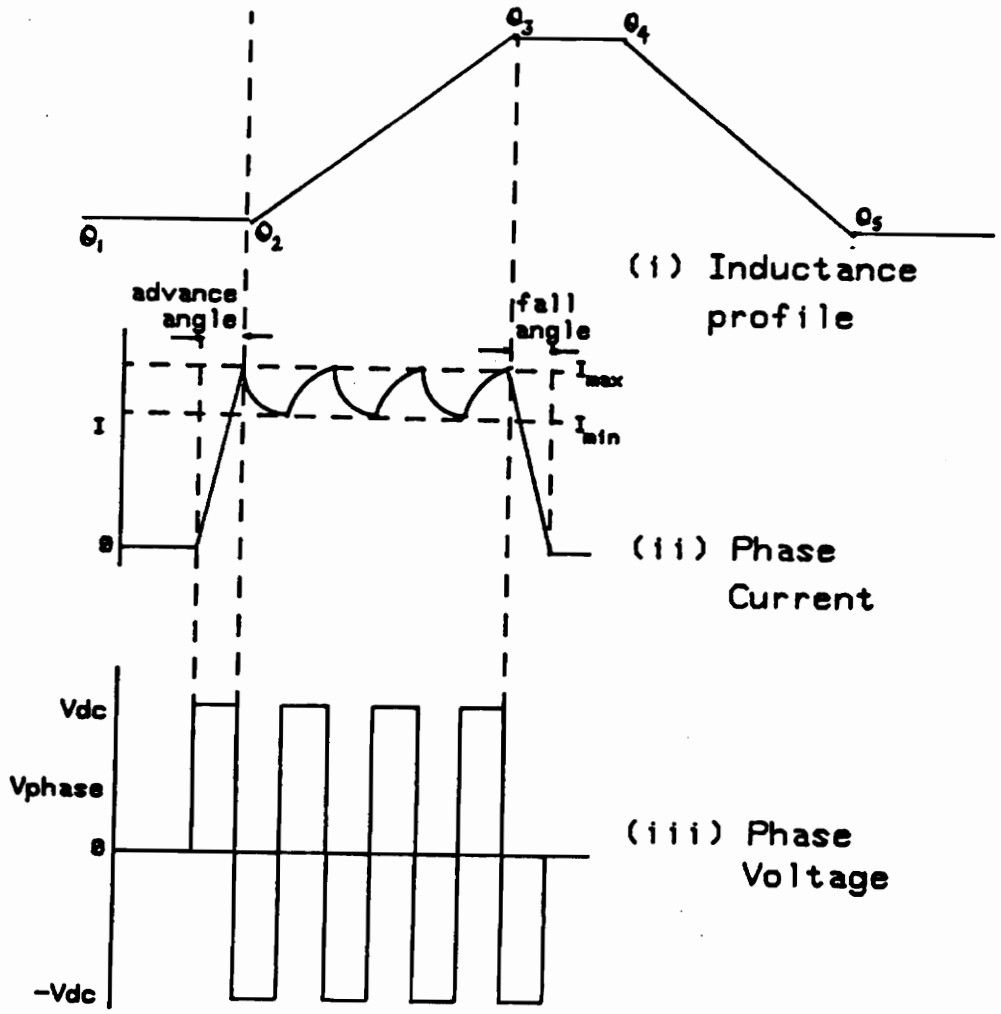


Figure 5. Inductance profile, current and voltage waveforms in a phase winding

advance angle, called the fall angle is difficult to estimate because the value of the inductance is not known at the instant of turning off. However, for low speeds and small currents, it can be approximated as

$$\theta_f = \frac{IL_{max}\omega_r}{V_d} \tag{2.8}$$

where L_{max} is the aligned inductance.

Further design considerations of the controller are discussed in the following sections. The block diagram of the closed loop SRM drive is shown in Figure 6 on page 17. A proportional integral (PI) controller is implemented to calculate the current command, I^* from the speed error $\Delta\omega_r$ [12]. The rotor position θ_r is used to calculate speed. The actual speed of the k^{th} sampling time is computed from θ_r using

$$\omega_{rk} = \frac{\theta_{rk} - \theta_{rk-1}}{\Delta t_k} \tag{2.9}$$

where θ_{rk} is the kth reading of the rotor position and Δt_k is the kth sampling period. Using the actual speed of the rotor and the speed command, the speed error $\Delta\omega_{rk}$ is calculated as

$$\Delta\omega_{rk} = \omega^*_{rk} - \omega_{rk} \tag{2.10}$$

where ω^*_{rk} is the commanded speed at the k^{th} sampling instant. The torque command, T^*_{ek}, is calculated using the PI control algorithm,

$$T^*_{ek} = k_p\Delta\omega_{rk} + k_i\sum_{j=1}^{k}\Delta\omega_{rj}\Delta t_j \tag{2.11}$$

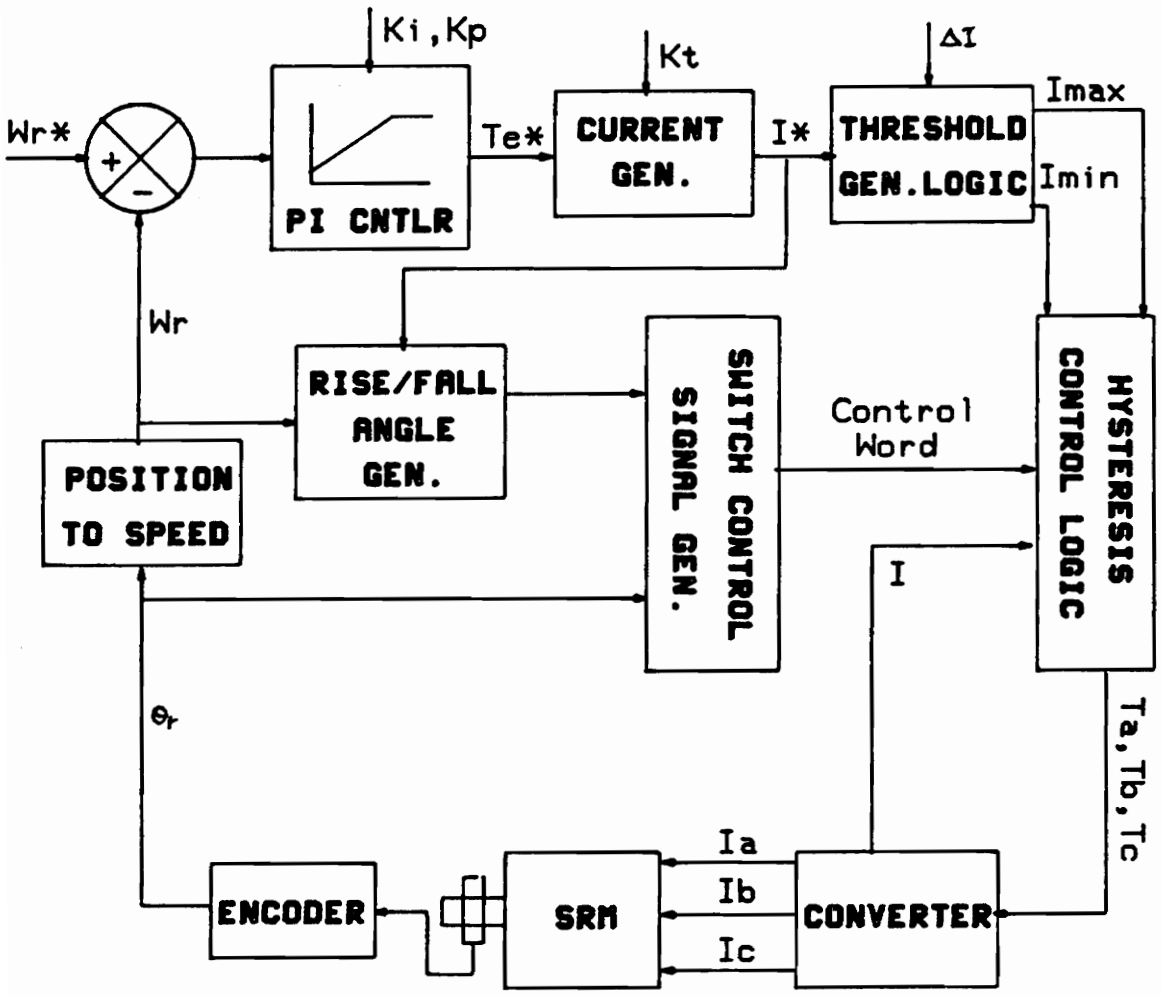


Figure 6. Closed loop SRM drive

where k_p is the proportionality constant and k_i is the integral constant. An easier approach to calculate the torque command is used in the software and is based on the algorithm,

$$T^*_{ek} = T^*_{ek-1} + \Delta T^*_{ek} \tag{2.12}$$

The incremental torque ΔT^*_{ek} is given by,

$$\Delta T^*_{ek} = k_1 \Delta \omega_{rk} - k_p \Delta \omega_{rk-1} \tag{2.13}$$

where

$$k_1 = k_p + \Delta t_k k_i \tag{2.14}$$

If the speed error exceeds a set limit, the integral control is cut off in the PI algorithm to improve the dynamics of the motor. This generally happens at the time of starting or if the commanded speed is changed. Once the speed error is reduced, the integral control is reintroduced.

From the torque command, the current command I^* is calculated as follows:

$$I^* = \sqrt{|T^*_e|/k_t} \tag{2.15}$$

where

$$k_t = \frac{1}{2}\frac{dL}{d\theta} \tag{2.16}$$

is the torque constant. It is dependent upon the magnitude of the current and the rotor position. The limits of the hysteresis window within which the current is maintained are calculated from I* using

$$I_{min} = I^* - \Delta I \tag{2.17}$$

$$I_{max} = I^* + \Delta I \tag{2.18}$$

where $2\Delta I$ is the hysteresis window. Once the current window is determined, it is given as an input to the hysteresis control logic. This logic block generates the switching signal based on the base drive control signals, commanded current and actual current. The phase switches of the active phases are turned off once the current reaches I_{max}. At this instant, regeneration takes place, the current decreases as the negative of the bus voltage appears across the winding. Once the current drops to I_{min}, the phase switches are turned on depending upon the base drive control signal.

The current flow in the motor is controlled by the switching signals T_a, T_b and T_c . The encoder fitted on the motor generates the absolute position of the motor which is used to calculate the speed, thus closing the loop. A switch control signal generation block generates the control signal based on the rise/fall angle and the rotor position.

2.3.1 PC as a controller

Digital controllers are becoming increasingly popular as controllers for motor drives. Some of the advantages of these controllers over analog controllers are that they can

be run under software control and provide higher level of communication with the user.

A microprocessor-based controller is an ideal choice for the implementation of a controller. The design of such a controller for a SRM drive has been discussed in various publications [10,11]. Many of the commercially available IC's like the timer, A/D converter, peripheral interface, etc., can be conveniently interfaced with the microprocessor to implement the necessary control features. But in order to provide adequate communication with the user, the microprocessor has to be interfaced with the monitor and keyboard through video controller cards and keyboard controller boards. This becomes a heavy burden on the designer of the controller. Hence, the most widely used microprocessor-based system, the Personal Computer, with its built in keyboard, monitor and disk interface and other library routines, becomes a very powerful tool as a controller for a motor drive.

The advantages of a PC based controller are as follows:

1. The PC processor itself can be used as the microprocessor.
2. The PC bus can be used to interface with a compatible Input/Output (I/O) board, thus making the I/O completely software controlled.
3. The drive status can be continuously updated and displayed on the monitor.
4. On line changes of parameters that profoundly affect the drive performance can be effected from the keyboard.
5. The built in library routines, called the Read Only Memory Basic Input Output System (ROM BIOS) can be used to perform some I/O functions, thus reducing the burden on the designer.

6. The DAS software can be integrated with the controller software to capture drive data for performance evaluation.

Due to the aforementioned advantages, the SRM controller and DAS have been implemented on the PC. The PC serves as a very sophisticated instrument to analyze the performance of the SRM drive which is in the research and development stage.

A general block diagram of the PC based controller is shown in Figure 7 on page 22. The PC is interfaced with the keyboard and monitor. Parameters such as θ_a, θ_f, ω_r^*, *etc.* , can be altered from the keyboard interactively. The motor speed and the current command are displayed on the monitor. An I/O board is interfaced with the PC through the PC bus. The inputs to the PC are the rotor position from the encoder and phase currents from current sensors. The outputs are the control signals to the phase switches. A detailed description of the hardware and software features of the controller are given in the chapter 3.

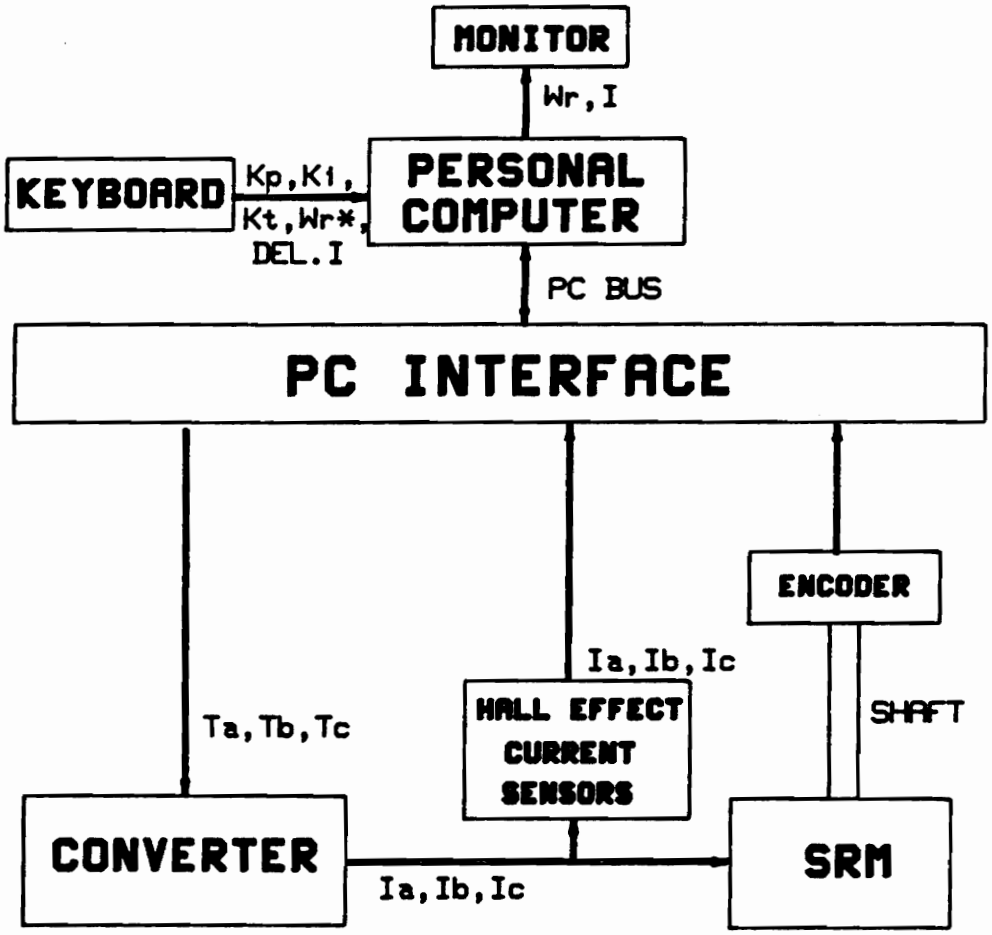


Figure 7. Block schematic of the PC based controller

3.0 Design of the PC based SRM controller

Chapter 2 discussed the advantages of the PC based controller and its configuration. This chapter presents a detailed design of the controller features. The controller functions are listed as follows:

1. Implementation of the PI control algorithm.
2. Tracing the position of the rotor to give the switching signals at the appropriate instants.
3. Implementation of the current loop using the hysteresis current controller.
4. Interfacing with the user through keyboard and monitor.
5. System protection by incorporating some safety measures.

The design of the controller can be further sub-divided into hardware and software. This chapter begins with a discussion on the software features of this controller. This is followed by a detailed description about the design of the PC interface board for the controller and the DAS.

3.1 Controller software

Since timing is very critical for this application, the software has been written in 8088 assembly language and has been implemented on an IBM PC running with a 4.77 MHz clock. Two important functions have to be implemented in software. Firstly, the control signals for the devices have to be generated based on the rotor position. Secondly, the PI algorithm has to be implemented based on the sampled values of the speed. Furthermore, the controller has to communicate with the external world. Hence, the software has been divided into three modules. They are

1. Switch signal generation routine
2. PI controller implementation routine
3. User interface module

3.1.1 Switch signal generation routine

This is an interrupt service routine which generates the appropriate switching signal depending upon the position of the rotor. This interrupt is generated by a counter which is toggled by the Least Significant Bit(LSB) of an absolute position encoder. Once the counter counts down to zero, this routine is executed. The software reads the current position from the encoder, determines the corresponding switching signal from a look-up table and sends it to the output port. Following this, the next switching instant is pre-determined. The switching instants are dependent upon the rotor position, θ_a and θ_f , and the direction of rotation. Based on the current position and the

above mentioned parameters, the next switching position is calculated. The difference between the present rotor position and the next switching position, divided by a factor of 2 is loaded into a count down counter. The factor of 2 arises from the fact that this counter is toggled by the LSB from the encoder and for every 2 bit change in position, the counter counts down by 1. The value N loaded into the counter is thus

$$N = [(\theta_{next} - \theta_{present})\frac{1024}{360}] / 2 \tag{3.1}$$

Since the encoder has a 10 bit output, 1024 corresponds to one revolution of the motor. The details about the interface hardware circuits are explained in the next section.

The software is also capable of generating switching signals for both overlapping and non-overlapping conduction angle between the two phases. Overlapping conduction is obtained when $\theta_a > \theta_f$ and non-overlapping conduction is obtained when $\theta_a < \theta_f$. The expression for determining the conduction angle for the SRM is

$$\theta_c = \frac{360°}{N_s N_r / 2} + \theta_a - \theta_f \tag{3.2}$$

where N_s is the number of stator poles and N_r is the number of rotor poles. However, during starting, the conduction angle should be at least 30° for the (6/4) SRM to avoid a gap between the switching signals of the two successive phases. Suppose the conduction angle is 25° and the position of the rotor from the reference point is, for example, 28° , there will be no firing pulse to any one of the 3 phases and the motor cannot be started.

The switching instants will completely change if the motor is in the braking mode, either due to a change in commanded speed which is less than the actual speed or due to a change in the direction of rotation command. In this mode, a negative torque is generated by turning on the devices during the falling inductance profile for each phase. In order to limit the magnitude of this torque to a certain value, the devices are turned on for a very short period, for example, 5° or 10°. This process is continued till the desired state is reached.

3.1.2 PI controller implementation routine

This is also an interrupt service routine and is executed every 4 milli-seconds(ms). In order to obtain a good performance, the PI controller is executed at a constant sampling interval which consequently simplifies the algorithm and reduces the execution time. The steps involved in implementing the PI control algorithm have been discussed earlier[12]. The factors which affect the implementation are discussed below.

Firstly, the sampling interval, Δt has to be decided. From a control point of view, it is better to have a small Δt as long as the computer has enough time for execution. But a small change in a variable will result in a large error. The position information is being used to compute speed and if, for example, the sampling interval is 2 ms, a 1 bit change in reading the encoder output will result in a speed error of

$$\frac{1/1024}{2 \times 10^{-3}} \times 60 = 30\,\text{rpm} \tag{3.3}$$

Considering the large mechanical time constant of the motor, a sampling interval of 4 ms is considered to be ideal.

Secondly, if the speed error is large, for example, during starting, it will result in a very large value of the commanded current, I^*. In order to protect the devices and the motor, the value of this current is limited to a safe value.

Thirdly, in order to maintain a constant torque at a given load, the current must be maintained constant. A hysteresis current controller is being used for this purpose. The PI algorithm decides a window $2\Delta I$ within which the current is maintained by the current controller. If this window is too small, the current will be maintained fairly constant but the devices will be stressed due to the high frequency switching. If the window is too large, the excursion of the current from the desired value will prevent the maintenance of a constant torque. Hence, an optimal window value is 5% - 10% of the commanded value.

3.1.3 User interface module

This part of the software takes the longest execution time but provides a user friendly interface through the keyboard and monitor. The most important function is to provide on-line modification of parameters such as θ_a and θ_t which affect the system performance. The commanded speed and direction of rotation can also be changed without having to stop the motor. These facilities greatly enhance the study of the drive in the research and development stage. A snapshot of the CRT display when the motor is running at 2500 rpm is shown in Figure 8 on page 29. The software also prevents the system from collapsing if an invalid parameter is entered by the user.

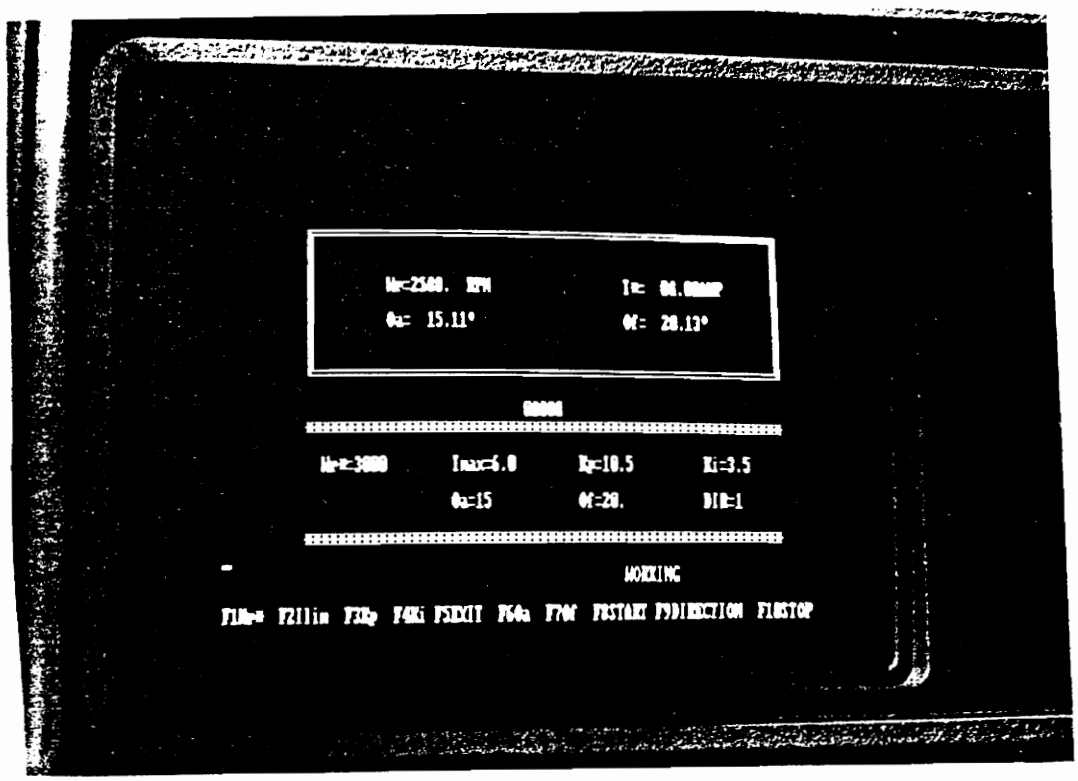


Figure 8. Display on the PC monitor

3.2 *Design of the interface board*

The design of the Printed Circuit Board (PCB) which serves as the interface between the IBM PC and the converter/motor is discussed in this section. This interface board has the following features:

1. Code converter
2. Three 16-bit timers
3. Two 8-bit and two 4-bit digital I/O ports
4. One 8-channel A/D converter
5. Two D/A converters
6. Hysteresis current controller

3.2.1 Code converter

The encoder that is being used to sense the position gives a ten bit digital value of the absolute position of the rotor. This ten bit value is in gray code format. Since it is very convenient to work with the binary format inside the computer and due to the fact that the LSB is necessary to clock one of the timers on board, it is essential to convert the gray code output from the encoder to binary format. This code conversion has been implemented using the 74LS86 exclusive-or gates. The conversion is implemented as shown in Figure 9 on page 31. The output of these gates is in ten bit binary format. However, since the PC being used is an IBM XT which has an eight bit external data bus, it is preferable to read an eight bit data for faster execution. This

problem is overcome by exploiting the symmetric configuration of the motor. Since this SRM is a (6/4) machine, the sequence of switching signals to the devices of each phase in the converter repeat every 90° rotation of the rotor. This corresponds to the least significant eight bits of the encoder output which is sufficient for the determination of the control signals.

3.2.2 Timer

Timers are used to keep track of time and generate interrupts so that the controller can perform specific functions. The 8253 Programmable Interval Timer which supports 3 16-bit timers is used for this purpose. The PI controller has been designed to run every 4 ms and uses timer 0 to generate interrupts. Timer 0 uses the system clock as its clock input, and interrupts on IRQ3 of the PC bus. The switch signal generation routine uses timer 1 to trace the position of the rotor so that the switching signals can be generated at the appropriate instants. Hence the LSB from the position encoder is used to clock it. This timer generates interrupts on IRQ2 of the PC bus. Timer 2 is used by the DAS to keep track of the time taken for measurements. Since the timing of the switching signals is very critical, timer 1 is given higher priority than timer 0.

3.2.3 Digital Input/Output

The 8255 Programmable Peripheral Interface(PPI) is used for digital I/O. It consists of two 8-bit ports and two 4-bit ports. The lower 4 bits of port C are used to output the

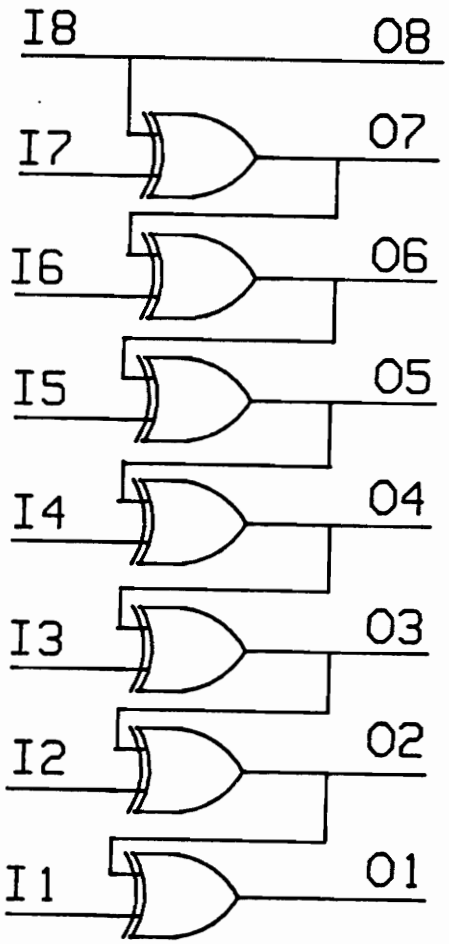


Figure 9. Conversion from gray code format to binary format

switching signals to the devices. Port B is configured to read the 8 bit position available from the code converter. The control signal of phase A is read as an input by the LSB of port A for use by the DAS.

3.2.4 A/D and D/A converters

An 8 channel A/D converter ADC0808 is used to perform A/D conversion. Three channels are used to read the phase currents. The A/D converter has a sampling frequency of 25KHz, which is good enough for this application.

Two high speed D/A converters, DAC08 are used to generate a signal proportional to the maximum and minimum values of the current window, I^*_{max} and I^*_{min} , respectively. An output voltage range of 0 to 5 volts corresponds to a commanded current range of 0 to 20 amperes. Moreover, the D/A converter does not have an inherent latching capability. Hence the inputs are latched using 74LS373 latches.

3.2.5 Hysteresis current controller

A hysteresis current control logic has been built to ensure that the phase current is maintained within preset limits. This circuit has been implemented using comparators, flip-flops and AND gates and is shown in Figure 10 on page 33. One of the comparator compares the phase current with I^*_{max} while the other one compares it with I^*_{min}. When the phase current exceeds I^*_{max}, the output of the lower comparator resets the flip-flop, thereby turning off the phase switch. When the phase current

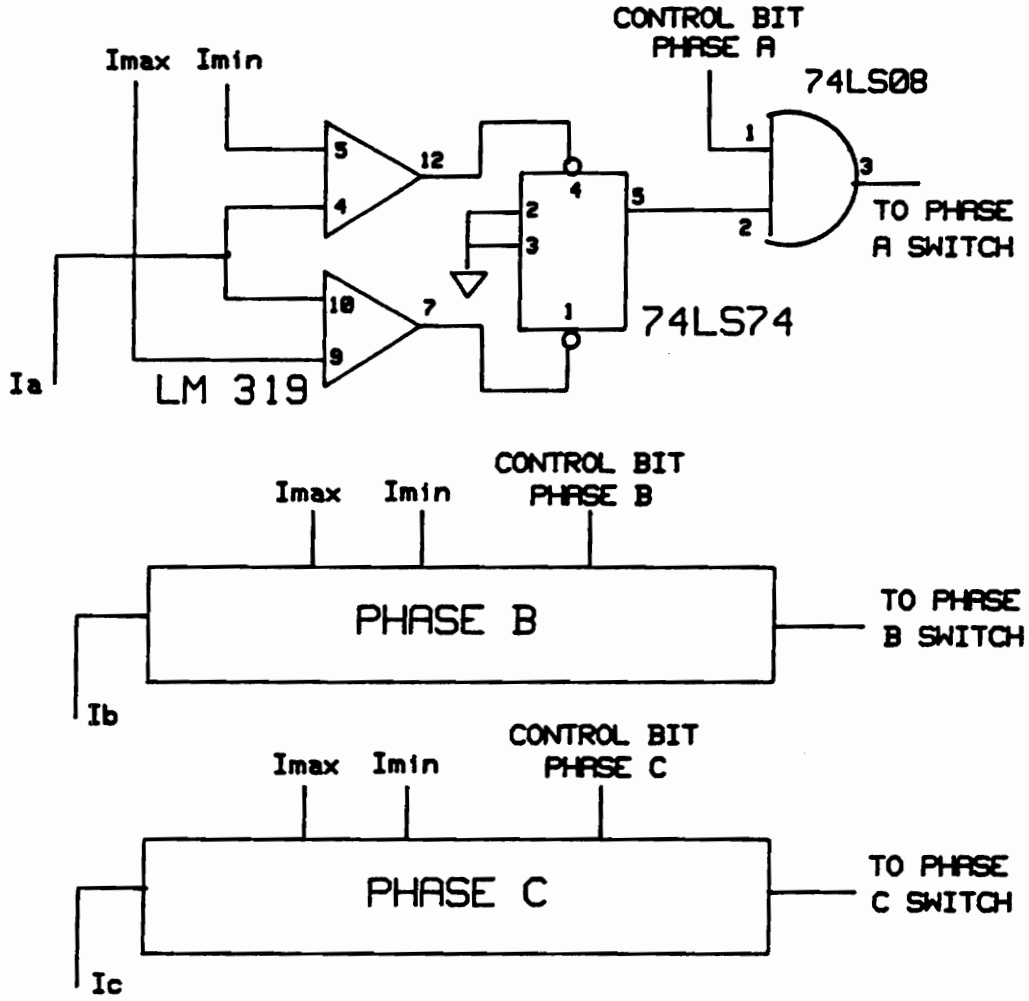


Figure 10. Implementation of hysteresis current controller

goes below I^*_{min}, the upper comparator presets the flip-flop, thus turning on the phase switch. There are three sets of current control logic, one per phase. The control signal for each switch from port C of the PPI is ANDed with the flip-flop output to ensure that the proper switch is selected.

The interface board has been successfully designed in the form of a PCB and tested. A schematic of the interface board is shown in Appendix A. A photograph of the PCB is shown in Figure 11 on page 35. The method of acquiring the data from the drive using this interface board is discussed in the next chapter.

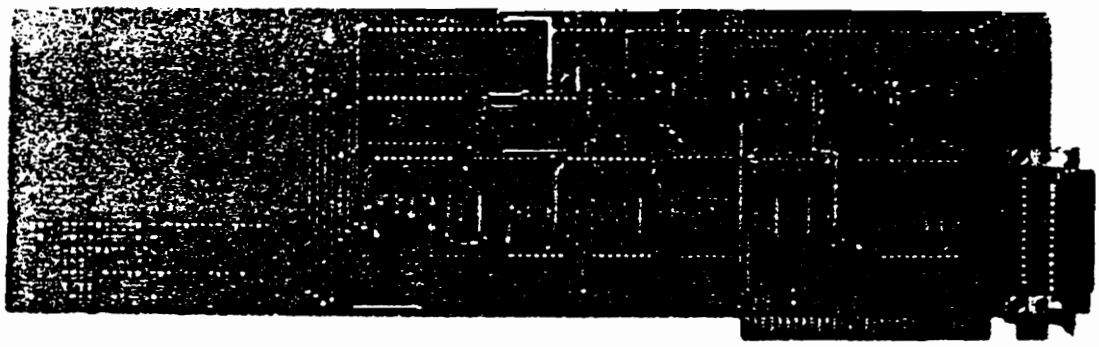


Figure 11. Photograph of the Printed Circuit Board

4.0 The Data Acquisition System

This chapter discusses the approach adopted for acquiring the motor parameters. Since the DAS can be operated only in conjunction with the controller, the first section discusses this interface. Time sharing between the DAS and the controller is also discussed. Finally, the machine parameters required for performance evaluation and the method of acquiring and storing these parameters is considered.

4.1 Interface with the SRM Controller

The DAS for the SRM cannot be a general purpose DAS due to the varied nature of the machine parameters needed for performance estimation. Hence, an application specific DAS, which is used for the evaluation of SRM characteristics has been designed and built. The DAS is interfaced with the controller in order to make more efficient use of the available resources and to make the system cost-effective. An interface between the PC and the motor/converter has been tested for the controller.

The hardware available on the interface board is sufficient to record the necessary parameters. Hence it is preferable to interface the DAS software with the controller software and use the same interface board for data acquisition.

The overall configuration of the system is shown in Figure 12 on page 38. The DAS inputs are acquired from the motor, converter and controller.

4.1.1 Analysis of timing

In order to acquire as much data as possible within the available time, the software has been written in 8088 assembly language. The data thus acquired is stored on a disk for later use by the performance evaluation software. This program has been written in C language.

To start with, the main program (data storage) calls the controller program. The necessary operating conditions are set by changing the appropriate parameters. Once the steady steady state operating condition is reached, the DAS is activated. The required number of points are acquired and control is returned to the controller. Once the controller is stopped, the main program receives the parameters from the DAS. The actual value is determined by using an appropriate multiplication factor and stored.

Let us consider the time available for the execution of the DAS and the controller. The PI controller routine takes 0.5ms, the control signal generation routine takes 0.2ms and the DAS takes 0.1ms for execution on a PC running with a 4.77MHz clock. The PI controller can be cut-off in the period during which the DAS is running. This is

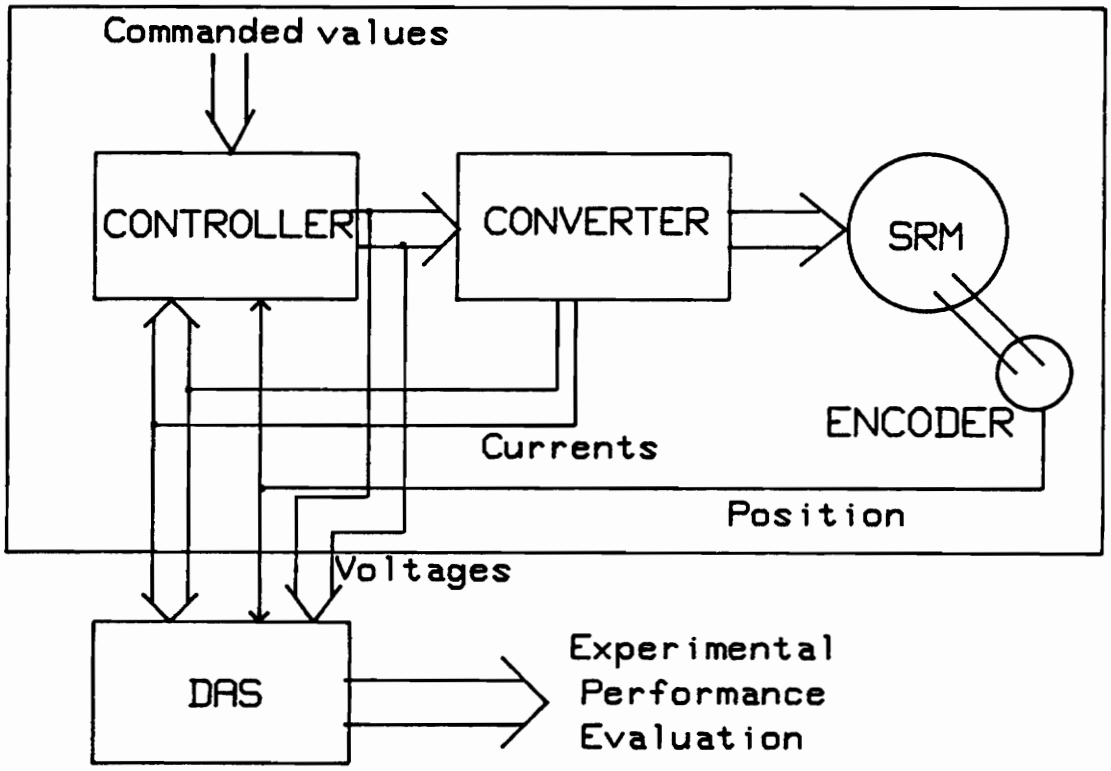


Figure 12. Overall Scheme of the DAS

because the system is already in the steady state when the DAS starts execution. The finite execution time of the DAS and the signal generation routine sets the maximum speed at which the motor can be run without loss of data during data acquisition. For example, at a speed of 1000 rpm, the motor takes 15ms for one fourth of the revolution. This is the period of interest to us since this corresponds to one full cycle for a particular phase as shown in Figure 5 on page 15. At this speed, measurements can be made for every 0.7° movement of the rotor (corresponding to a 2-bit change in the encoder output). The rotor takes about 0.125ms to rotate through 0.7°. Since the DAS executes in a lesser time span, no loss of data should occur. But some data points are still missed due to the execution of the signal generation routine. Since the motor has to run continuously, this routine has been given the facility to interrupt the DAS.

4.2 Parameters necessary to be acquired

The very first consideration in the data acquisition process is the period of time over which data should be logged that would be sufficient to estimate the performance of the machine. For steady state evaluation it is sufficient to record the data over one period of time in the inductance profile. For the (6/4) SRM, this period is 90° which is shown as θ_5 in Figure 5 on page 15. It is also sufficient to record the data set for one of the phases.

The different kinds of data that needs to be recorded has to be determined. Torque can be determined from the current and inductance of the corresponding position.

Speed is calculated from the time taken for a 90° rotation of the motor. Output power is calculated from torque and speed. Input power is computed from phase voltage and phase current. Hence the following parameters are recorded.

1. **Time.** Time taken between successive data points is calculated from a timer running continuously over the period of measurement.
2. **Position.** The position is recorded as a digital value from the position encoder using a digital I/O port.
3. **Voltage.** The phase voltage can be conveniently determined from the switching signal and the MOSFET and diode voltage drop as shown in Figure 13 on page 41, if the bus is maintained stiff. When the phase switch is on, the phase voltage is

$$V_{ph} = V_{dc} - 2I_aR_{ds} \tag{4.1}$$

where R_{ds} is the on-state resistance of the MOSFETs. Since two MOSFETs in series conduct when a phase is on, a factor of two is included. If the phase switch is off and there is a current flow in the winding, the phase voltage is

$$V_{ph} = -V_{dc} - 2V_d \tag{4.2}$$

where $2V_d$ is the drop across two diodes during regeneration.

4. **Current.** The current is amplified by a current amplifier and measured from a A/D converter.

The flow chart for the DAS is shown in Figure 14 on page 43. Once the data acquisition command is given, the interrupt for the PI controller is disabled. The measure-

Ia	T1/T2	Va
↑	1	Vdc−2Vs
↓	1	Vdc−2Vs
↓	0	−Vdc−2Vd
0	0	0

Figure 13. Table for the determination of phase voltage from switch signal

ment takes place when the position is 45°. The timer is then initiated for continuous counting. 128 data points are recorded when the speed is less than 1000 rpm. The measurement is made for every 2 bit change in the encoder output, which corresponds to a 0.7° rotation of the rotor. The switch signal generation routine interrupt is disabled for a small time during which measurements are initiated. This is to ensure that all the measurements of a given set are done at the same instant. This does not cause any change in the performance of the machine because even in the worst case the switching instants are altered only by a few tenths of a degree. Once all the data is acquired, control is transferred to the controller.

After the motor is stopped, the main program retrieves the data from the DAS. Before the data is stored, it has to be converted to the actual value as follows:

1. The user is prompted to enter the bus voltage during acquisition. This, in conjunction with Figure 13 on page 41 is used to calculate the actual phase voltage.
2. The current amplifier has been preset to give an output of 0 to 5 volts for a current range of 0 to 20 A. The range of the 8-bit A/D converter is 0 to 255. Hence the actual current is determined as

$$I = \frac{X}{255} \times 20 \tag{4.3}$$

   where X is the A/D converter output.
3. The position encoder gives a range of 0 to 255 for a 90° rotation of the motor. Hence, the actual value of the position in degrees is calculated as

$$\theta = \frac{Y}{256} \times 90° \tag{4.4}$$

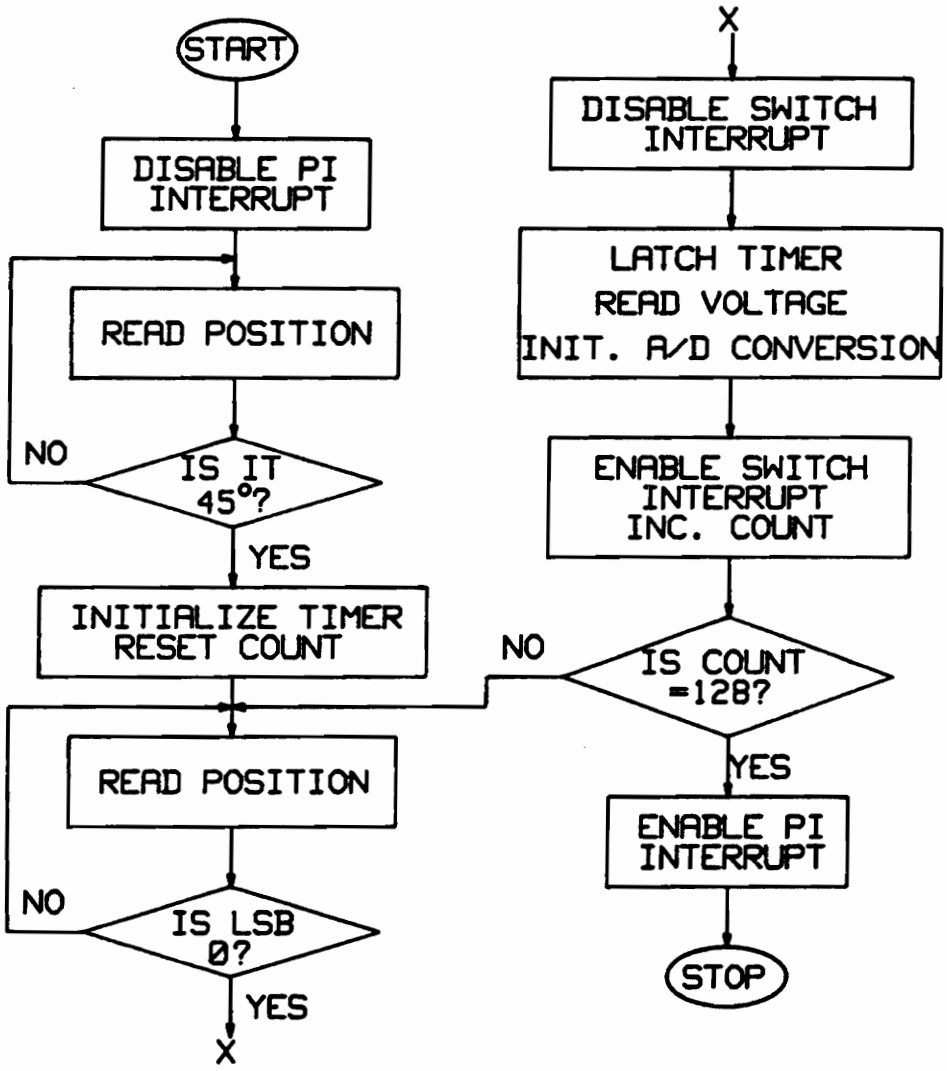


Figure 14. Flowchart for the Data Acquisition Software

where Y is the encoder output.

4. The count value of the timer has to be converted to the actual value in μ sec. Since the timer is clocked by a 4.77MHz clock, the time is obtained as

$$t = \frac{Z}{4.77} \times 10^{-6} \tag{4.5}$$

where Z is the timer output.

Since the timer has a 16-bit output, it rolls over after a count of 65,536. Care is taken to account for this overflow when the time is calculated.

Finally, when all the data is calculated in the appropriate form, it is stored in a disk for use by the performance evaluation software.

5.0 Performance Evaluation

This chapter discusses the evaluation of performance parameters using the data acquired. Torque and torque ripple are calculated from the current and inductance at the corresponding position. Using the speed and average torque, developed power is calculated. The input power is determined from the phase voltage and phase current. With the computation of losses, the output power and efficiency are also determined.

5.1 Prediction of torque and torque ripple

The calculation of torque is explained with reference to Figure 15 on page 47. This figure shows the flux linkages characteristics for three different positions. The inductances have been measured from 0° to 45° in steps of 5° Thus, the ψ vs. current characteristics are determined for various positions. The inductance profile for a particular phase has a period of 90°. The profile from 0° to 45° is identical to the pro-

file from 90° to 45°. Hence all the positions in the 90° to 45° range are mapped to corresponding positions in the 0° to 45° range.

The workdone in moving the rotor from one position to the next has to be determined. This is interpreted as the change in mechanical energy and is given as [16]

$$\Delta W_m = \Delta W_e - \Delta W_f \tag{5.1}$$

where ΔW_e is the change in electrical energy and ΔW_f is the change in the stored magnetic field energy. Referring to Figure 15 on page 47, if the rotor moves from θ_1 to θ_2, ΔW_e is given as the area AGHCA and ΔW_f is given as the difference between area OHCO and area OGAO. Hence, ΔW_m is

$$\begin{aligned} \Delta W_m &= area(AGHCA - OHCO + OGAO) \\ &= area\ OACO \end{aligned} \tag{5.2}$$

Similarly, ΔW_m is calculated when the rotor moves from θ_2 to θ_3 and so on. An easier approach is used to calculate ΔW_m in the software. If W_1 is the area under the curve (i.e. area between the curve and the x axis) for the first position and W_2 is the area under the curve for the second position, then

$$\Delta W_m = W_1 - W_2 \pm trap\_area \tag{5.3}$$

Trap_area is either subtracted or added depending upon whether the current is increasing or decreasing, respectively. For example, if the rotor moves from θ_1 to θ_2, the trap_area ABDCA is subtracted from $(W_1 - W_2)$. If θ_1 is the fully unaligned position, the curve is linear, and W_1 is given as

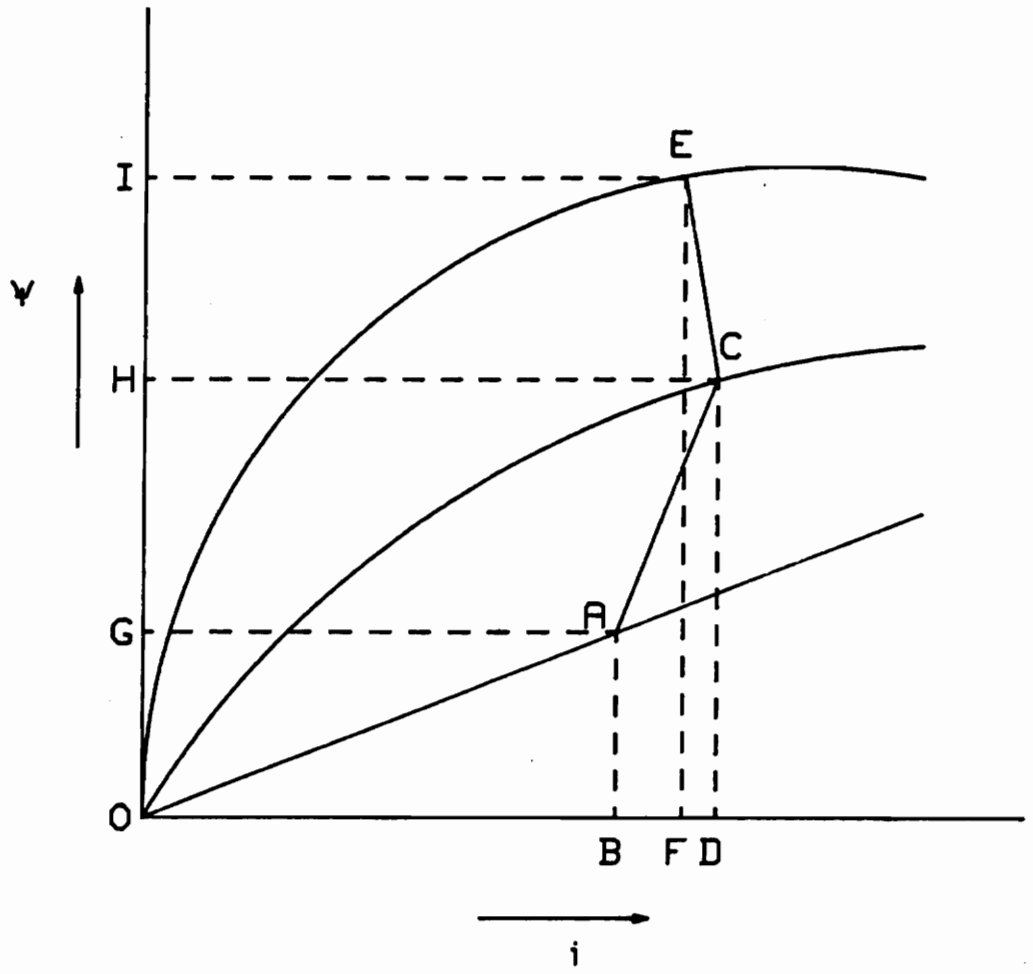


Figure 15. Flux linkage characteristics

$$W_1 = \frac{\psi_A i_A}{2} \tag{5.4}$$

If the curve is non-linear the trapezoidal method is used to calculate the area under the curve and it is given as

$$W_2 = \frac{1}{2}\Delta i_1 \psi_1 + \Delta i_2 \psi_2 + \Delta_3 i_3 + \ldots + \frac{1}{2}\Delta i_n \psi_n \tag{5.5}$$

This method is accurate enough if many points are considered on the curve. Since the current step is 0.25A up to a current of 5A and 0.5A beyond that, the error is minimized. The Simpson's integration method can be used if more accuracy is needed.

The instantaneous torque is determined for one of the phases for which data has been acquired. The instantaneous torque is given as

$$\begin{aligned} T_{inst} &= \frac{\Delta W}{\Delta \theta} \\ &= \frac{\text{Workdone}_{(\theta_1 \text{to} \theta_2)}}{|\theta_2 - \theta_1|} \end{aligned} \tag{5.6}$$

with reference to Figure 15 on page 47.

The torque developed by the other two phases are identical, except for the fact that they are displaced by 30° and 60° respectively from this phase. Hence the torque developed by the other two phases are superimposed on this torque with the appropriate displacement. The total instantaneous torque over a 90° period is then determined by adding up the torque generated by the individual phases. The average

torque developed by the motor is determined by using numerical integration over one time period.

$$T_{av} = \frac{1}{T}\sum_{i=1}^{n} T_{inst}\Delta t_n \tag{5.7}$$

where n is the total number of data points acquired by the DAS, T is the time period(90° rotation) and Δt_n is the time difference between two successive data points.

The torque ripple is determined by doing a fourier analysis of the instantaneous torque. Since the torque function has been discretized into a sequence of N samples, discrete fourier transform(DFT) is applied. The DFT of a function f(t) sampled Δt time units apart is

$$\begin{aligned} F(u) &= \frac{1}{N}\sum_{t=0}^{N-1} f(t)\exp[-j2\pi ut/N] \\ &= \frac{1}{N}\sum_{t=0}^{N-1} f(t)[\cos(2\pi ut/N) - j\sin(2\pi ut/N)] \end{aligned} \tag{5.8}$$

for u = 0, 1, 2, ... N-1. F(u) is of the form A + jB. The fourier spectrum is obtained from the magnitude of each of the transform terms; that is

$$|F(0)| = \sqrt{A_0^2 + B_0^2}$$

$$|F(1)| = \sqrt{A_1^2 + B_1^2} \tag{5.9}$$

$$|F(2)| = \sqrt{A_2^2 + B_2^2}$$

and so on. $|F(0)|$ gives the average value of the torque function. The remaining terms give the magnitude of the harmonics at that particular frequency. The ripple content in the function is calculated as

$$\text{ripple} = \frac{\sqrt{|F(1)|^2 + |F(2)|^2 + \ldots + |F(N-1)|^2}}{|F(0)|} \tag{5.10}$$

5.2 *Estimation of losses*

The rms value of the current is calculated as

$$i_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} i_n^2} \tag{5.11}$$

when the sampling interval remains constant. If the resistance/phase of the motor is R_{ph} , the copper losses are

$$P_{cu} = 3i_{rms}^2 R_{ph} \tag{5.12}$$

The calculation of stator and rotor core losses involves complex analytical methods and has been dealt with in literature[9]. In this experimental set-up, the total core losses are determined as

$$P_{core} = P_{in} - P_{cu} - P_d \tag{5.13}$$

where P_{in} is the power input and P_d is the power developed, and their calculation is discussed in the next section.

Windage and friction losses (P_{fw}) are a function of speed. P_{fw} has been determined for different speeds for the test set-up and is shown in Figure 16 on page 52. Since the speed of the motor is known, P_{fw} is easily determined.

The converter losses are calculated depending upon the mode of operation, the motoring mode or the regeneration mode. However, the switching losses are neglected. During motoring, the losses occur in the MOSFETs and the instantaneous loss is given as

$$P_{conv} = 2I_a^2 R_{ds} \tag{5.14}$$

During regeneration, the converter loss is

$$P_{conv} = 2V_d I_a \tag{5.15}$$

The average converter loss is then calculated from the instantaneous losses.

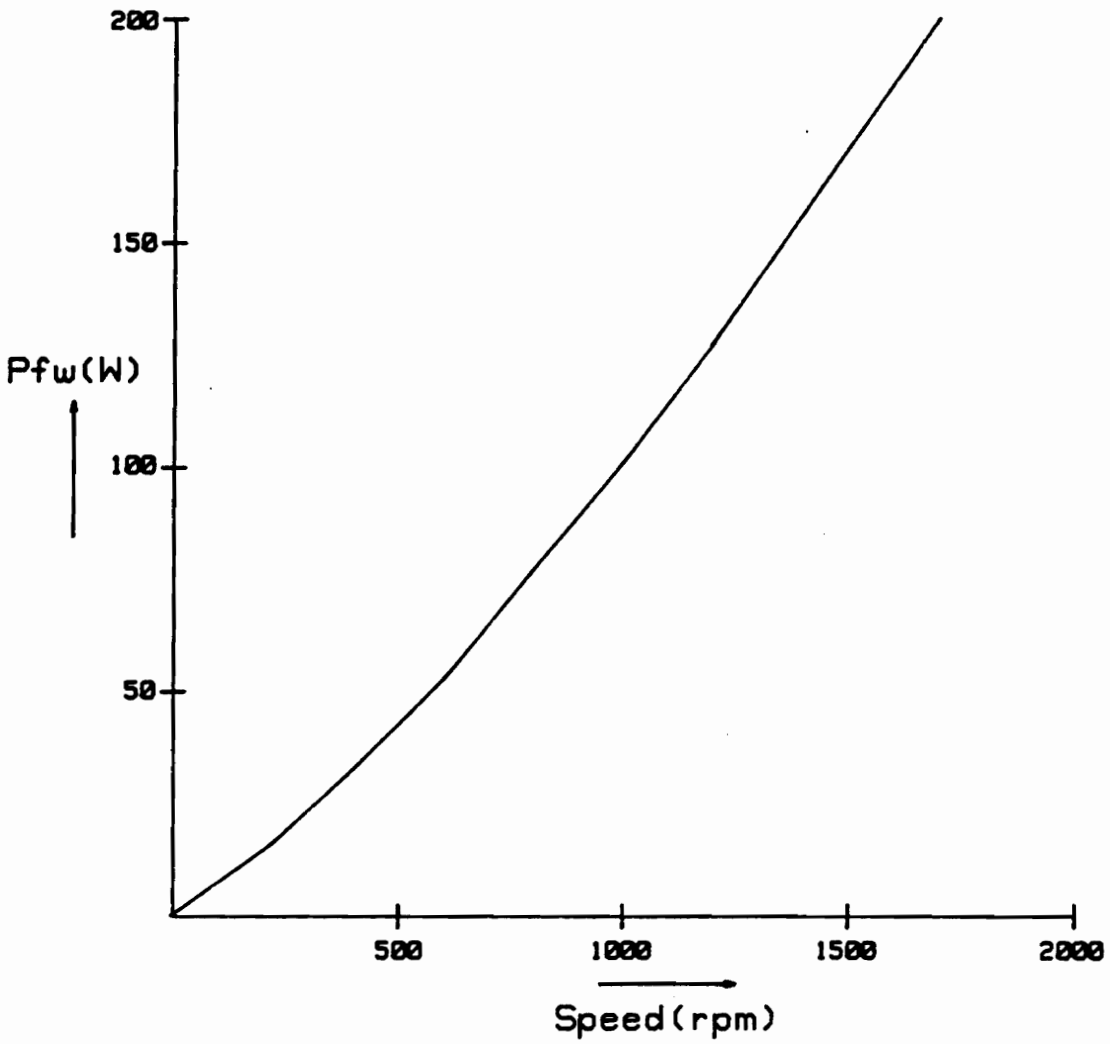


Figure 16. Windage and Friction losses for the SRM

5.3 Input and Output Power

The input power is determined from the phase voltage and phase current. If V_{phase} is the instantaneous voltage and I_{phase} is the instantaneous current, the instantaneous power/phase is calculated as

$$P_{inst} = V_{phase} \times I_{phase} \tag{5.16}$$

The average power input is

$$P_{in} = \frac{1}{T} \sum_{i=1}^{n} P_{inst} \Delta t_n \tag{5.17}$$

where T is the total time of acquisition (90°period) and Δt_n is the time difference between two successive data points. Since the input power is the same for all the three phases, the total power input is obtained by multiplying P_{in} by 3.

The motor speed is determined from the time taken to rotate the rotor through 90° . The time taken by the motor for a complete revolution, t_{360} is then computed. Hence, the motor speed in rpm is

$$n = \frac{60}{t_{360}} \tag{5.18}$$

The power developed by the machine is given as

$$
\begin{aligned}
P_d &= T_{av} \times \omega \\
&= \frac{WN_sN_r}{4\pi} \times \frac{2\pi n}{60} \\
&= \frac{WN_sN_rn}{120}
\end{aligned}
\tag{5.19}
$$

where W is the mechanical workdone by the machine in moving from the unaligned position to the aligned position and back to the unaligned position. The output power is

$$
P_{out} = P_d - P_{fw} \tag{5.20}
$$

and the efficiency is

$$
\eta = \frac{P_{out}}{P_{in}} \tag{5.21}
$$

6.0 Results and Discussions

This chapter discusses the experimental results obtained from the prototype SRM drive using the DAS. The results are compared with the measured data from the dynamometer and the probable cause for the error is also discussed.

6.1 *Experimental Results*

The DAS has been used to acquire data from the drive at different speeds and loads. Figure 17 on page 57 shows the performance data of the drive for one particular load. θ_a was 0°, θ_f was 8.08° and the load current was 5A. The motor was programmed to run in direction 0. The data was calculated using the algorithms presented in chapter 5. Figure 18 on page 58 to Figure 20 on page 60 show the phase current, phase voltage and input power as a function of time. Figure 21 on page 61 to Figure 23 on page 63 show the same set of data as a function of rotor position. Since θ_a was set to 0°, it can be seen that the current flow starts when $\theta_r = 60°$(fully aligned position

for phase A is 0°). The input power closely follows the current, since the phase voltage is nearly constant. The direction of power flow depends upon the state of the phase switches. Around 82° , when the switches are turned off, the motor regenerates and power is fed back to the supply. The phase voltage also changes sign depending upon whether the mode of operation is motoring or regeneration. Figure 24 on page 64 depicts the instantaneous torque per phase. The torque is a function of current and $\frac{dL}{d\theta}$. Figure 27 on page 67 shows inductance vs. rotor position for three different currents. It can be seen that there is a sudden change in slope at two different positions. This, in turn, produces a sharp change in the instantaneous torque although the current remains roughly constant. Since there is a current flow in the negative inductance region, a small negative torque is generated between 6° and 8° . The total instantaneous torque developed by all the 3 phases over a 90° period is given in Figure 25 on page 65. The torque developed by each phase is spaced apart by 30° from the other two phases. The fourier spectrum of the instantaneous torque is shown in Figure 26 on page 66. It gives the average value of the function and the weighted magnitudes of the harmonics.

Referring to Figure 17 on page 57, it can be seen that the efficiency of the motor is only 40.6%. This is because the load is small and the motor is running at a low speed. Figure 28 on page 68 shows the efficiency curve for the motor as a function of output power. The efficiency increases as the output power increases.

The performance evaluation software has been written in 'C'. The measured inductance profile and the data acquired from the drive serve as inputs to the program. The output of the program is the performance data for the particular set of in-

```
************************************************************
Electromagnetic torque  =   2.54 Nm
Power Developed         =   64.5 W
Torque ripple           =  104.5 %
Motor speed             =    243 rpm
Input Power             =  114.6 W
Output power            =   46.5 W
Efficiency              =   40.6 %
RMS phase current       =   1.21 A
Copper loss             =   15.9 W
Core loss               =   34.3 W
Windage loss            =   18.0 W
Converter loss          =    7.6 W
************************************************************
```

Figure 17. Performance data for one particular load

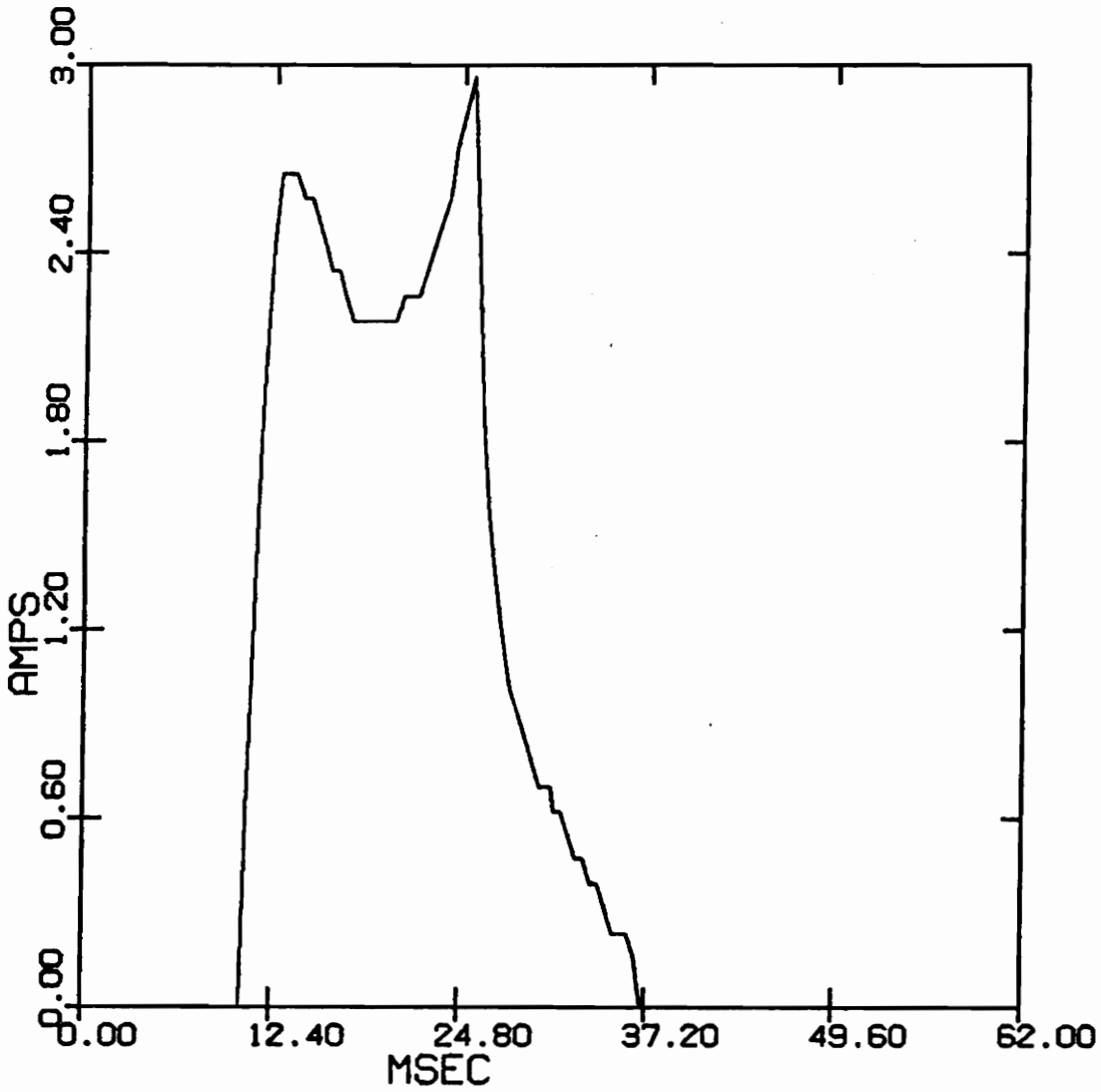


Figure 18. Phase current vs. time

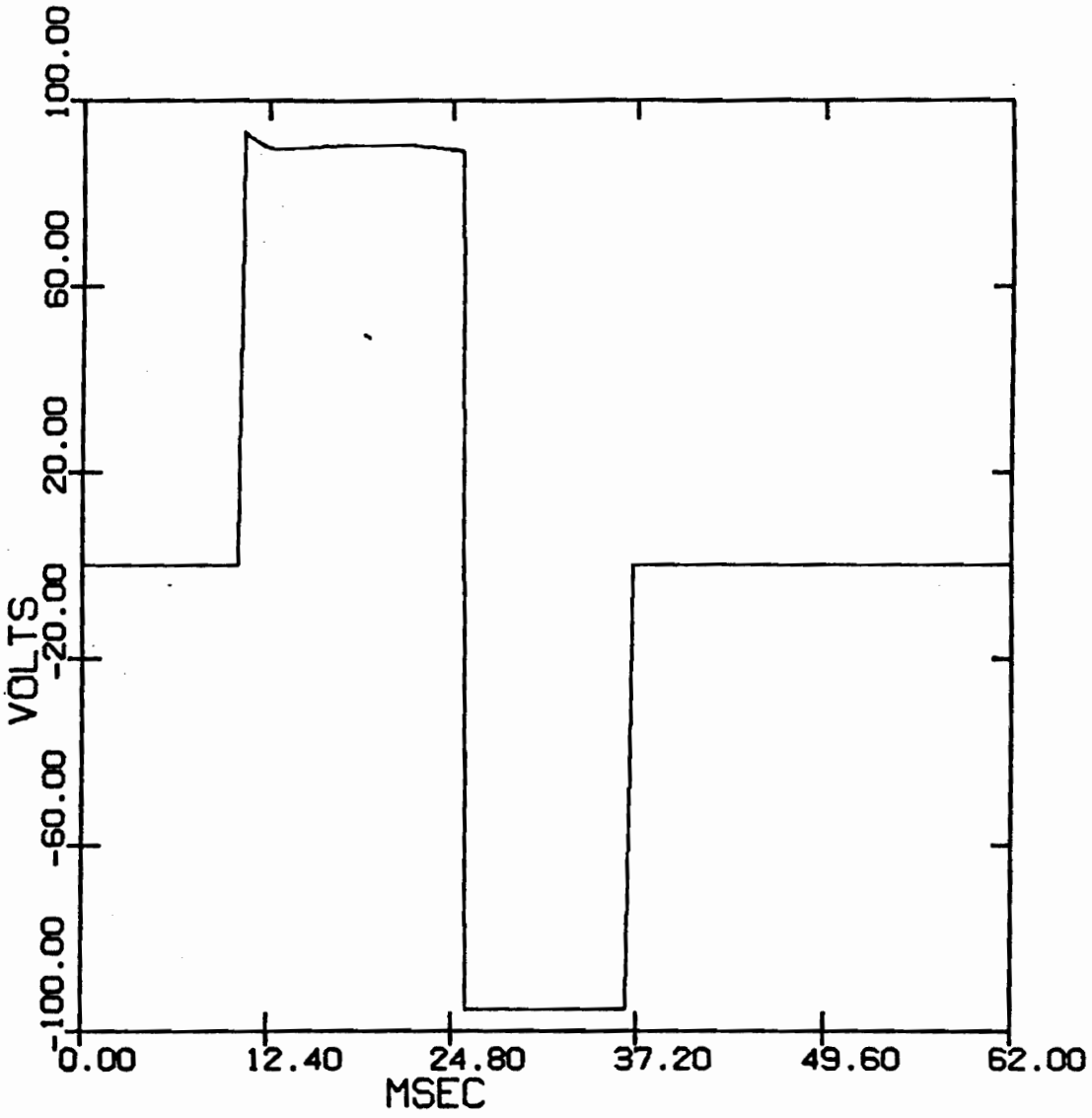


Figure 19. Phase voltage vs. time

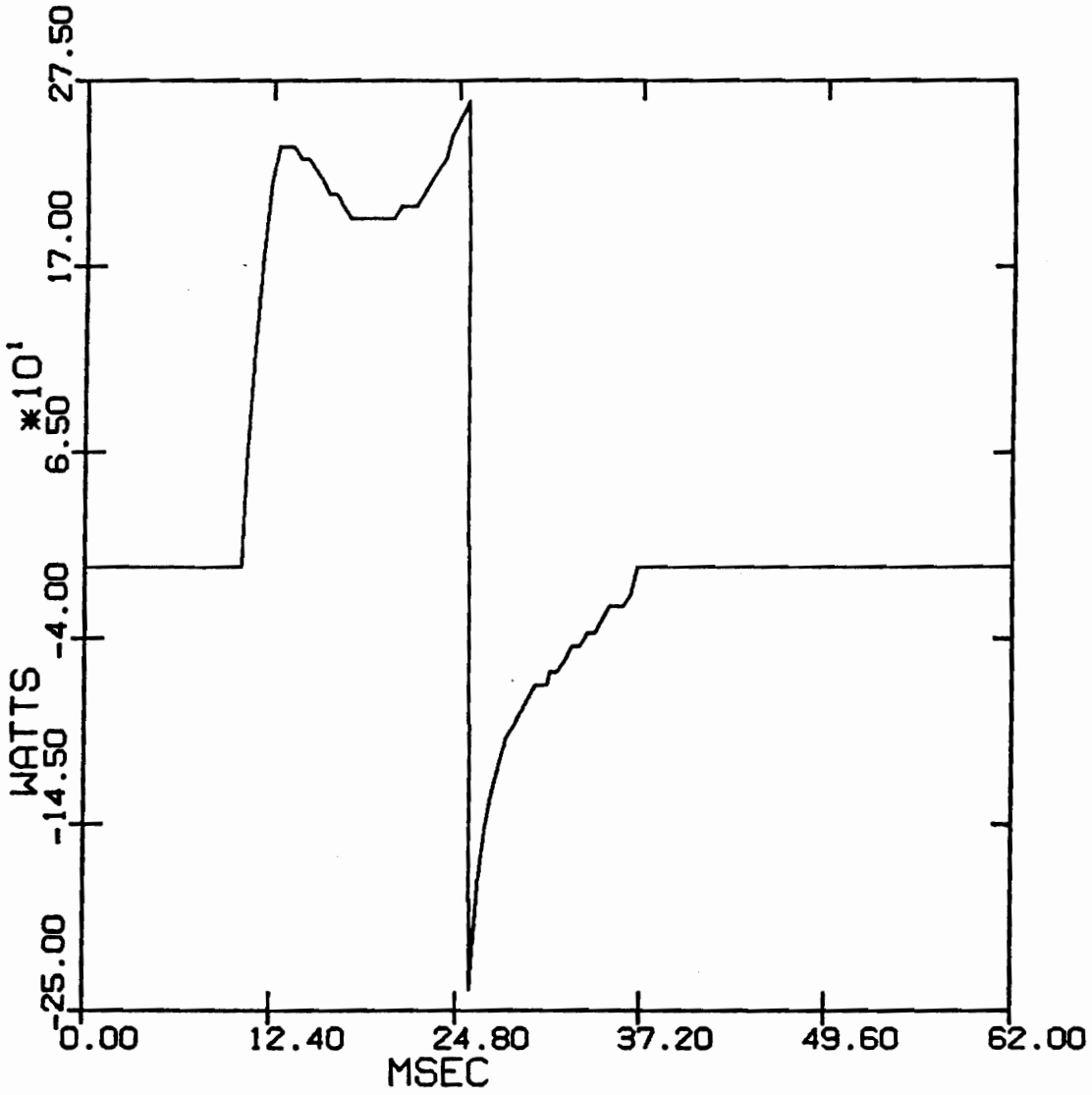


Figure 20. Input power/phase vs. time

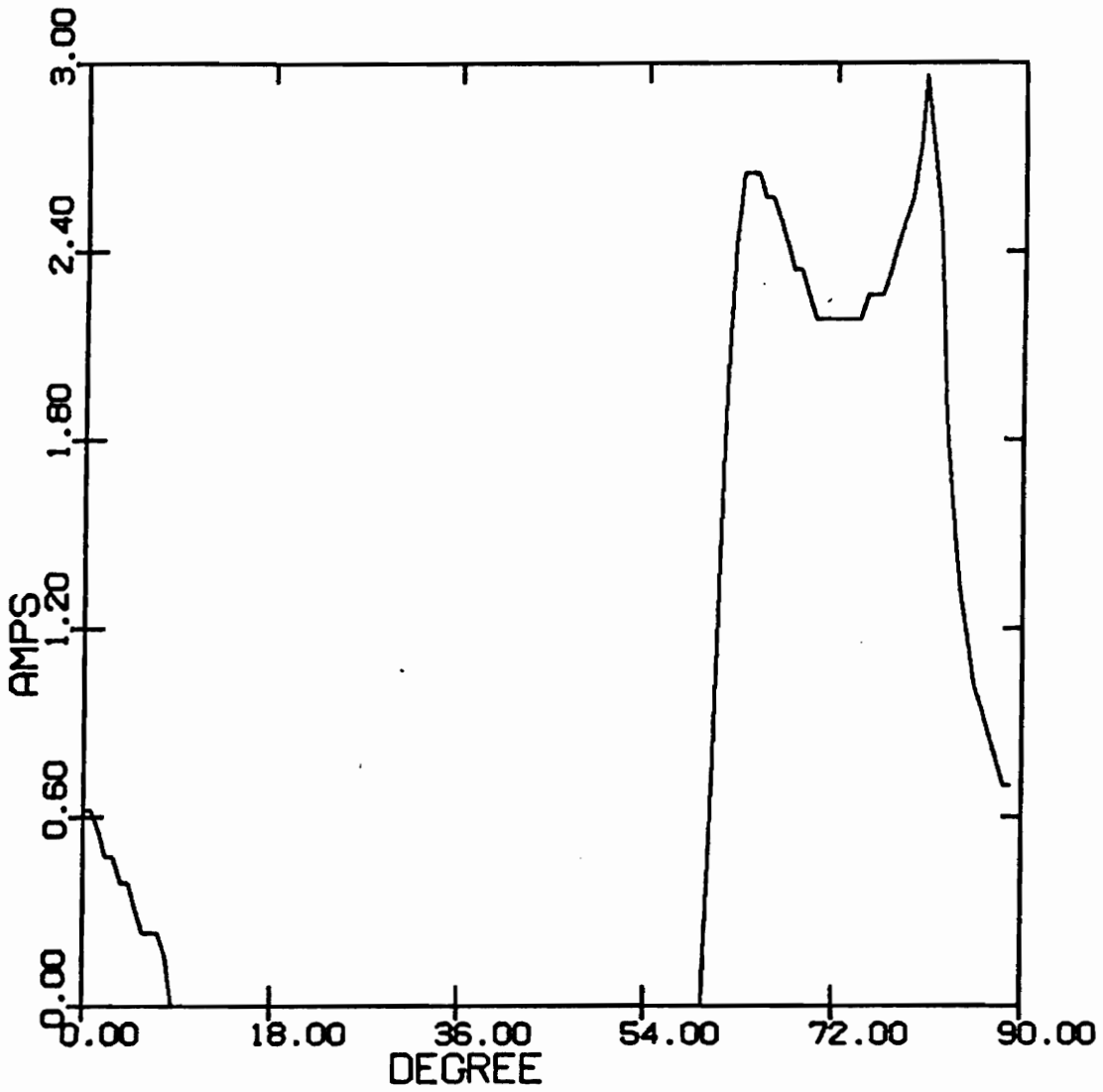


Figure 21. Phase current vs. rotor position

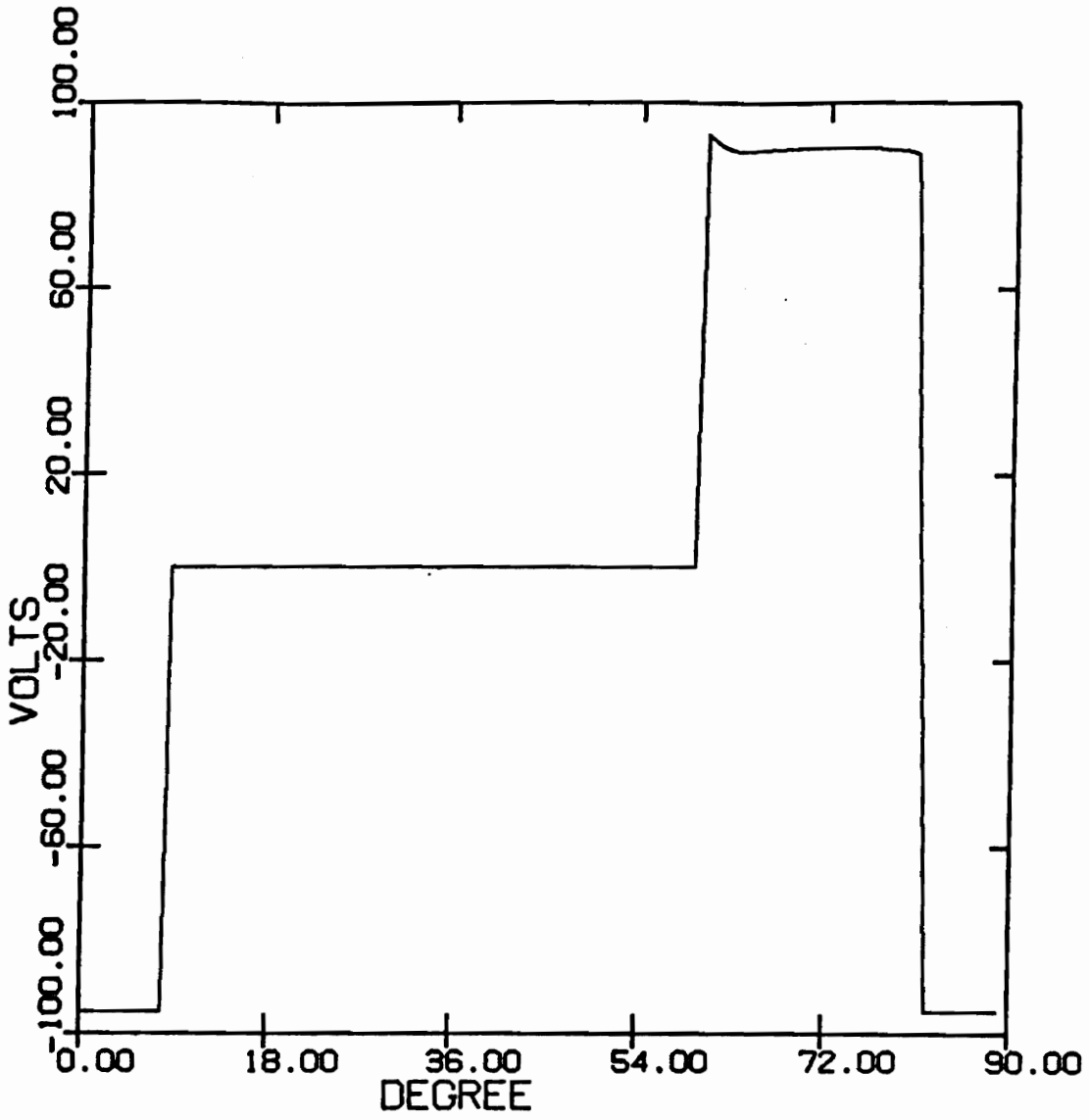


Figure 22. Phase voltage vs. rotor position

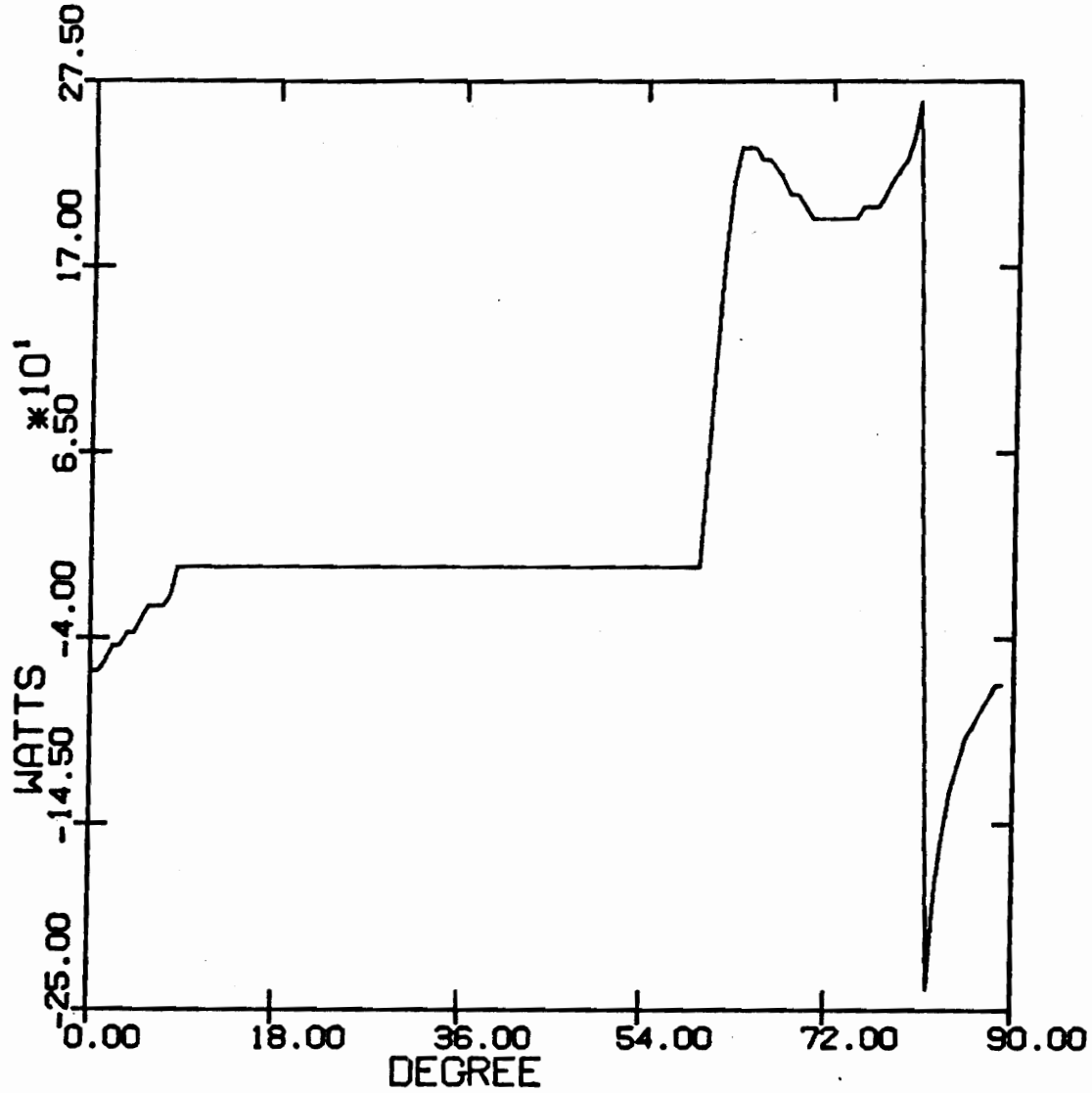


Figure 23. Input power/phase vs. rotor position

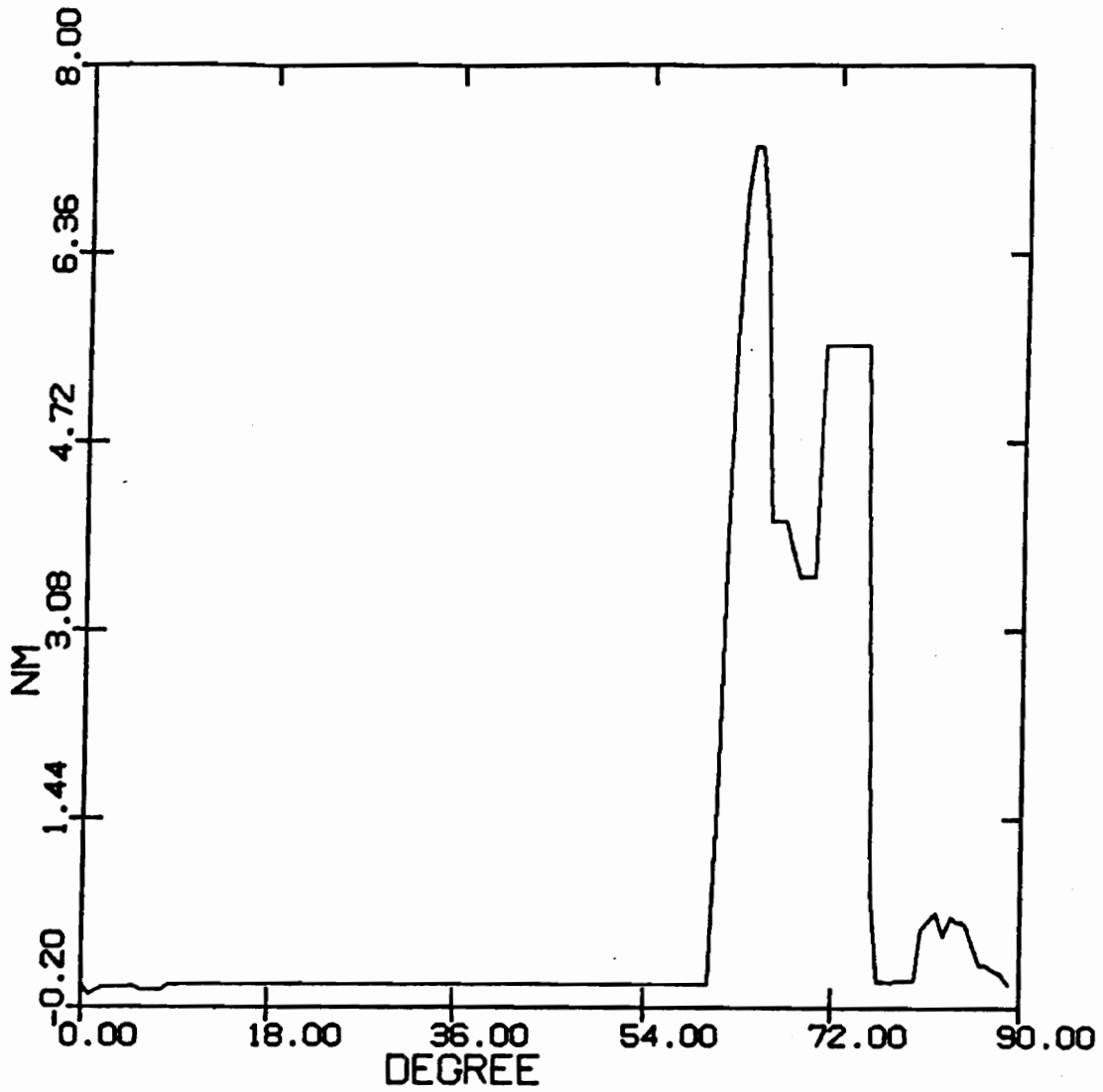


Figure 24. Instantaneous torque/phase vs. rotor position

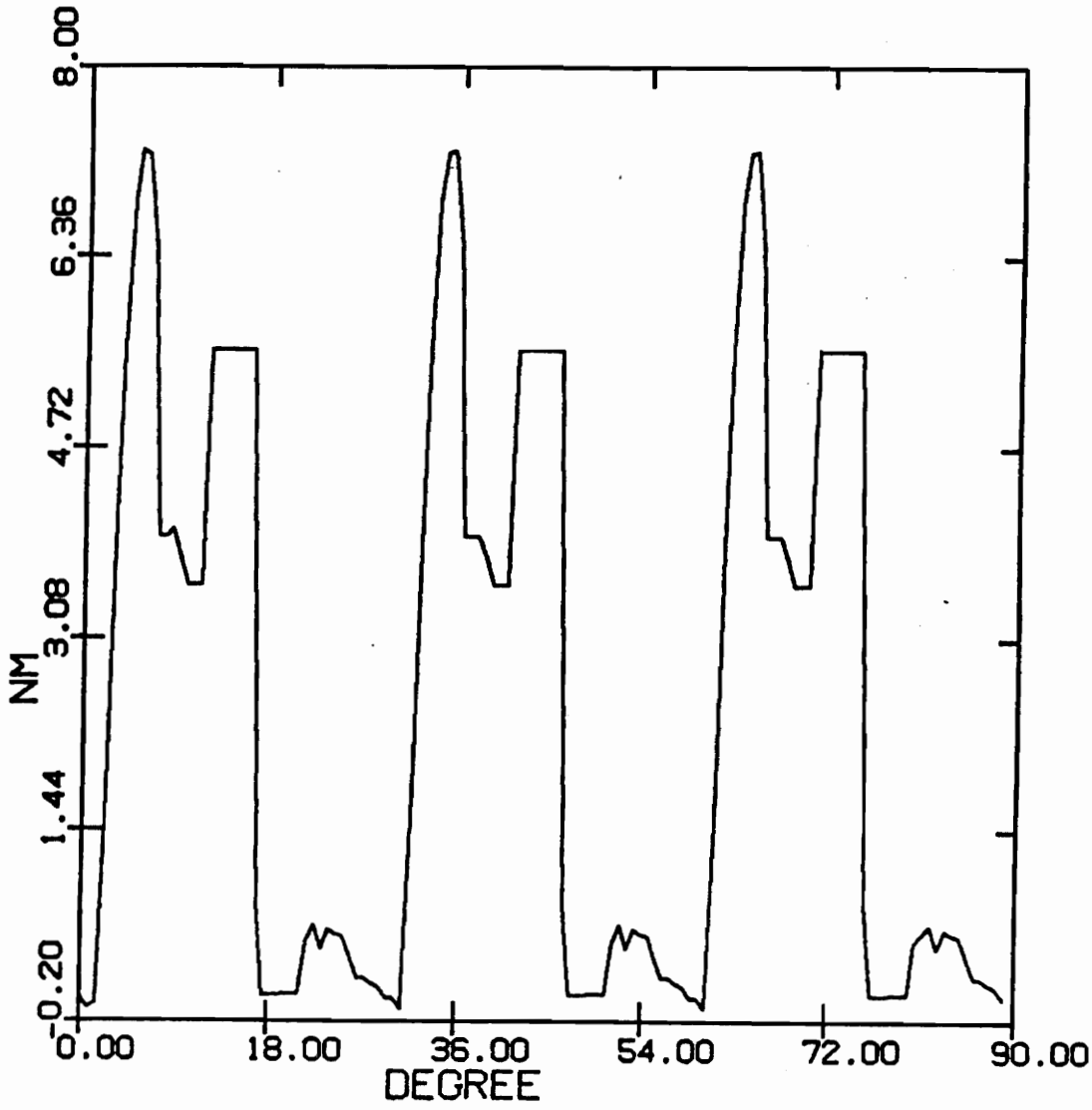


Figure 25. Total instantaneous torque vs. rotor position

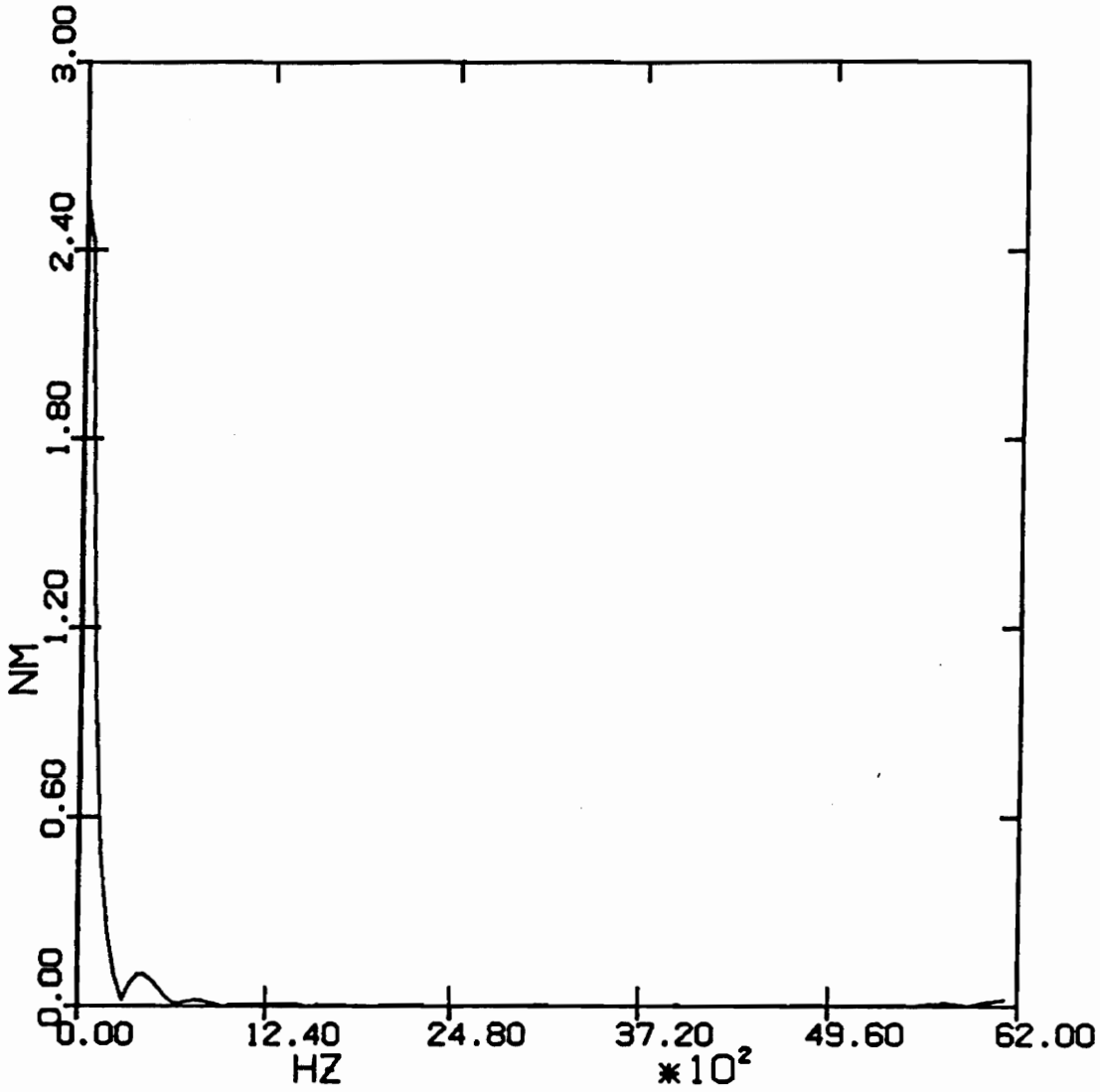


Figure 26. Fourier spectrum of the instantaneous torque

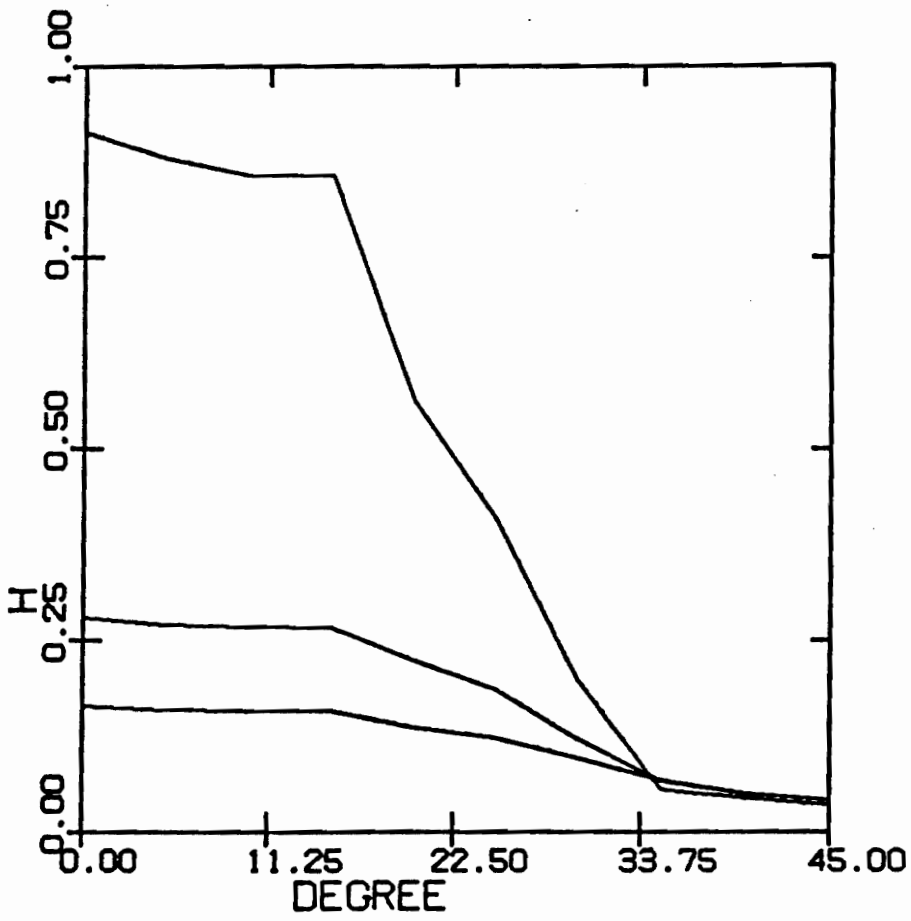


Figure 27. Inductance vs. rotor position for three currents

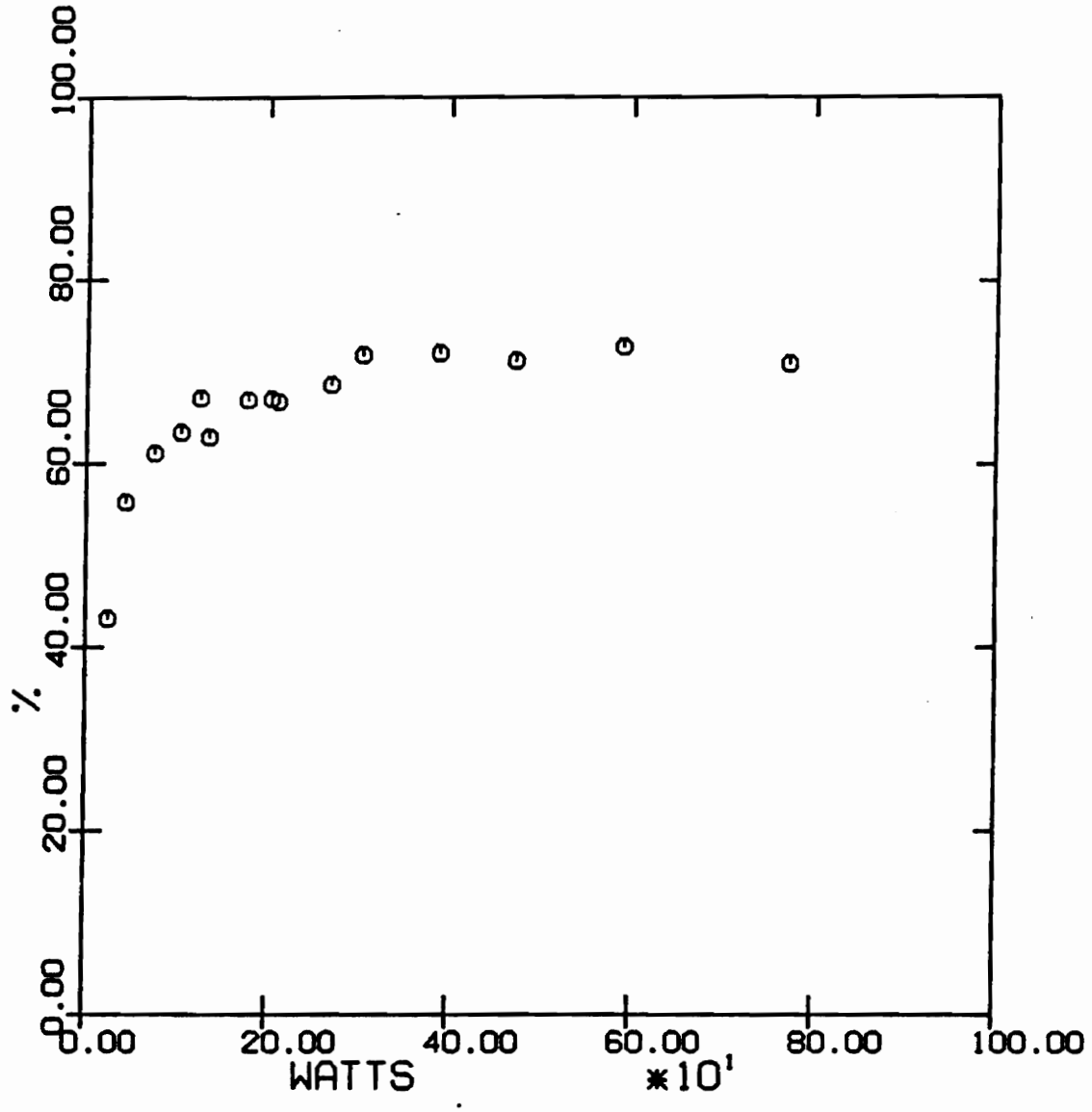


Figure 28. Efficiency vs. Output power

Vdc	Idc	Vdyn	Idyn	Speed	Dir	P(cal)	P(meas)	% Error
57	0.45	14.8	1.0	300	1	46.6	38.2	-22.0
66	0.55	14.6	2.0	303	1	66.1	54.1	-22.2
80	0.70	14.0	4.0	300	1	97.9	85.5	-14.5
89	0.85	13.5	6.0	300	1	127.7	118.5	-7.8
100	0.95	13.0	7.9	304	1	158.2	150.9	-4.8
106	1.10	12.7	10.2	309	1	201.1	195.0	-3.1
113	1.20	12.7	12.0	319	1	236.1	234.7	-0.6
130	1.30	14.0	14.0	354	1	297.5	302.1	+1.5
180	0.45	51.0	0.66	1000	1	159.6	135.1	-18.1
140	0.65	29.9	3.0	615	1	180.2	148.0	-21.7
168	0.85	28.4	6.0	605	1	256.5	238.3	-7.6
190	1.02	27.8	9.0	607	1	358.4	336.4	-6.5
201	1.20	26.4	12.1	594	1	440.3	430.3	-2.3
214	1.40	25.8	14.8	597	1	524.8	522.2	-0.5
230	1.60	25.7	18.0	611	1	644.4	646.3	+0.3
252	1.95	24.6	24.2	615	1	826.7	884.2	+6.5
94	0.65	11.0	5.0	243	0	64.5	83.0	+22.3

Figure 29. Comparison with measured data

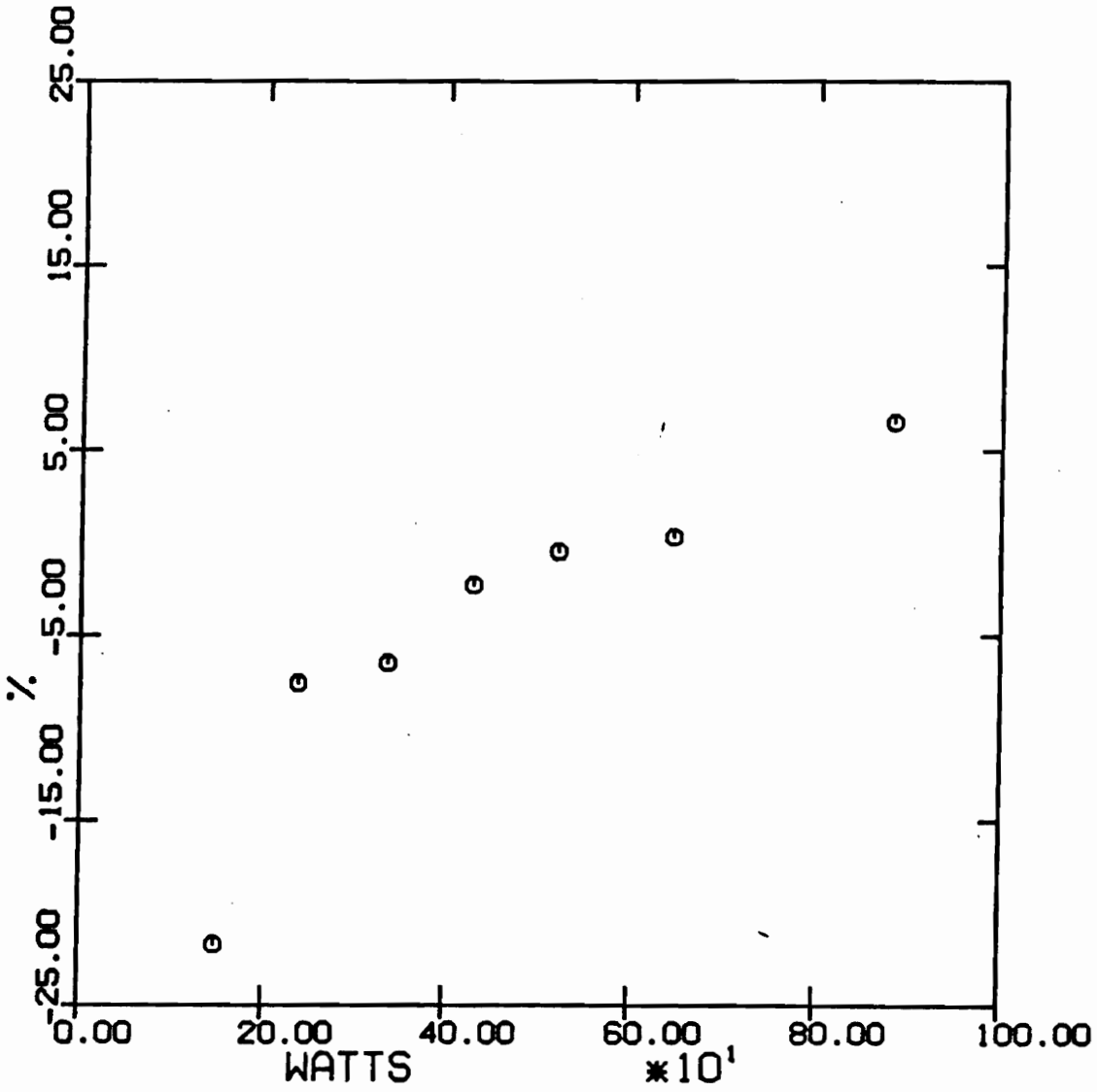


Figure 30. Error in the predicted results at a constant speed .

put. An user friendly plotting routine has been included to plot the performance characteristics of the motor.

6.2 Comparison with measured data

Figure 29 on page 69 tabulates the calculated data and the measured data for different loads. Figure 30 on page 70 graphically represents the error in the predicted results when the drive is running at 600 rpm with different load conditions. The parameters shown are

V_{dc} - Bus voltage, V

I_{dc} - Phase current, A

V_{dyn} - Load voltage, V

I_{dyn} - Load current, A

Speed - Motor speed, rpm

Dir - Direction of rotation

$P_{(cal)}$ - Power Developed (calculated), W

$P_{(meas)}$ - Power Developed (measured), W

Error - Difference between measured power and calculated power.

$P_{(meas)}$ is determined as

$$P_{(meas)} = V_{dyn} \times I_{dyn} + I_{dyn}^2 R_{dyn} + P_{fw} \quad (6.1)$$

where $R_{dyn}$is the dynamometer resistance and P_{fw} is the friction and windage losses at that speed. The error is calculated as

$$\text{Error} = \frac{P_{(meas)} - P_{(cal)}}{P_{(meas)}} \times 100 \qquad (6.2)$$

The predicted values are higher than the measured values at smaller loads. As the load increases, the error reduces. The probable causes for the error are as follows:

1. The inductances used for the calculation of torque does not match accurately with the actual inductances of the machine at small currents. The measured inductance had lot of transient data for currents less than 2 amperes. This will have a severe impact on the predicted results at smaller values of peak stator current. As the peak stator current increases, the error in the prediction of torque at smaller currents becomes negligible.

2. Since the inductances are known for positions in steps of 5°, extrapolation of flux linkages for intermediate positions will cause an error in the calculation of mechanical workdone.

3. The trapezoidal method of calculating the workdone will cause a small error in determining the area enclosed between two curves.

4. The approximation of the actual current to the nearest current for which inductance is known will also result in a small error.

7.0 Conclusions and Recommendations

Chapter 6 discussed the experimental results obtained from the prototype SRM drive using the DAS and the comparison with the measured data. This chapter outlines the contributions of this thesis. It concludes with the limitations of this implementations and the recommendations for future work.

7.1 Contributions

This thesis has attempted to develop an user friendly data acquisition system for the steady state performance evaluation of the switched reluctance motor drive. The DAS has been integrated with the SRM controller in order to make the system cost-effective. Performance parameters like electromagnetic torque, output power and efficiency have been predicted. The instantaneous torque and power have also been determined to get a better understanding of the machine. The predicted results have been compared with the dynamometer test results. The error in prediction is high at

small loads and tolerable at high loads. The probable causes for the error are also discussed.

7.2 Recommendations

One of the limitations of this DAS is that it cannot acquire the data under dynamic operating conditions. Since the speed of the machine is varying, the PI control routine has to be in effect. With the present configuration, this routine takes 0.5 msec for execution. Hence, only a few data points can be recorded with the current set-up under dynamic conditions. This will introduce a considerable error in the performance evaluation. In order to overcome this problem, two approaches can be mentioned.

1. The controller and the DAS can be implemented on a PC running with a faster processor and clock, like the IBM AT. An AT running with a 10 MHz clock will speed up the execution by at least 5 times, thus making data acquisition possible under dynamic conditions and higher speeds.

2. The DAS can be run on a different PC with a separate data acquisition board, thus making it completely independent of the controller. This solution, however, makes the drive very expensive.

8.0 Bibliography

1. Lawrenson P.J., et al, *"Variable Speed Switched Reluctance Motors,"* IEE Proc., Vol.127, Pt.B, No.4, pp 253-265, July 1980.

2. Krishnan R., Arumugam R. and Lindsay J.F, *"Design Procedure for Switched Reluctance Motors,"* Conf.Record, IEEE IAS Annual Meeting, Colorado, pp 858-863, Oct 1986.

3. Arumugam R., Lindsay J.F and Krishnan R., *"A Comparison of the Performance of Two Different Types of Switched Reluctance Motors"* Electric Machines and Power Systems, Vol. 12, pp 281-286, 1987.

4. Krishnan R., Aravind S. and Materu P., *"Computer Aided Design of Electrical Machines for Variable Speed Applications,"* Proc., IEEE IECON, pp 756-763, Nov 1987.

5. Krishnan R., *"Analysis of Electronically Controlled Machines,"* Class Notes, EE Dept., VPI&SU, Blacksburg, VA, 1986.

6. Davis R.M., Ray W.F. and Blake R.J., *"Inverter Drive for Switched Reluctance Motor: Circuits and Component Ratings,"* , IEE Proc., Vol 128, Pt.B, No.2, pp 126-136, Mar 1981.

7. Miller T.J.E., *"Converter Volt-Ampere Requirements of the Switched Reluctance Motor Drive,"* Conf. Record, IEEE IAS Annual Meeting, pp 813-819, 1984.

8. Materu P., Krishnan R. and Farzanehfard H., *"Steady State Analysis of the Variable Speed Switched Reluctance Motor Drive,"* Proc., IEEE IECON, pp 294-302, 1987.

9. Materu P. and Krishnan R., *"Estimation of Switched Relucance Motor Losses,"* Conf. Record, IEEE IAS, Pittsburgh, Oct 1988.

10. Bose B.K., et al, *"Microcomputer Control of a Switched Reluctance Motor,"* Proc., IEEE IAS, pp 542-547, 1985.

11. Oza A., Krishnan R., and Adkar S., *"Microprocessor based Controller for SRM motor drive,"* IEEE IECON, pp 458-453, Nov 1987.

12. Mang X., Krishnan R., Adkar S. and Chandramouli G., *"Personal Computer based Controller for Switched Reluctance Motor Drives,"* Proc., IEEE IECON, pp 550-555, 1987.

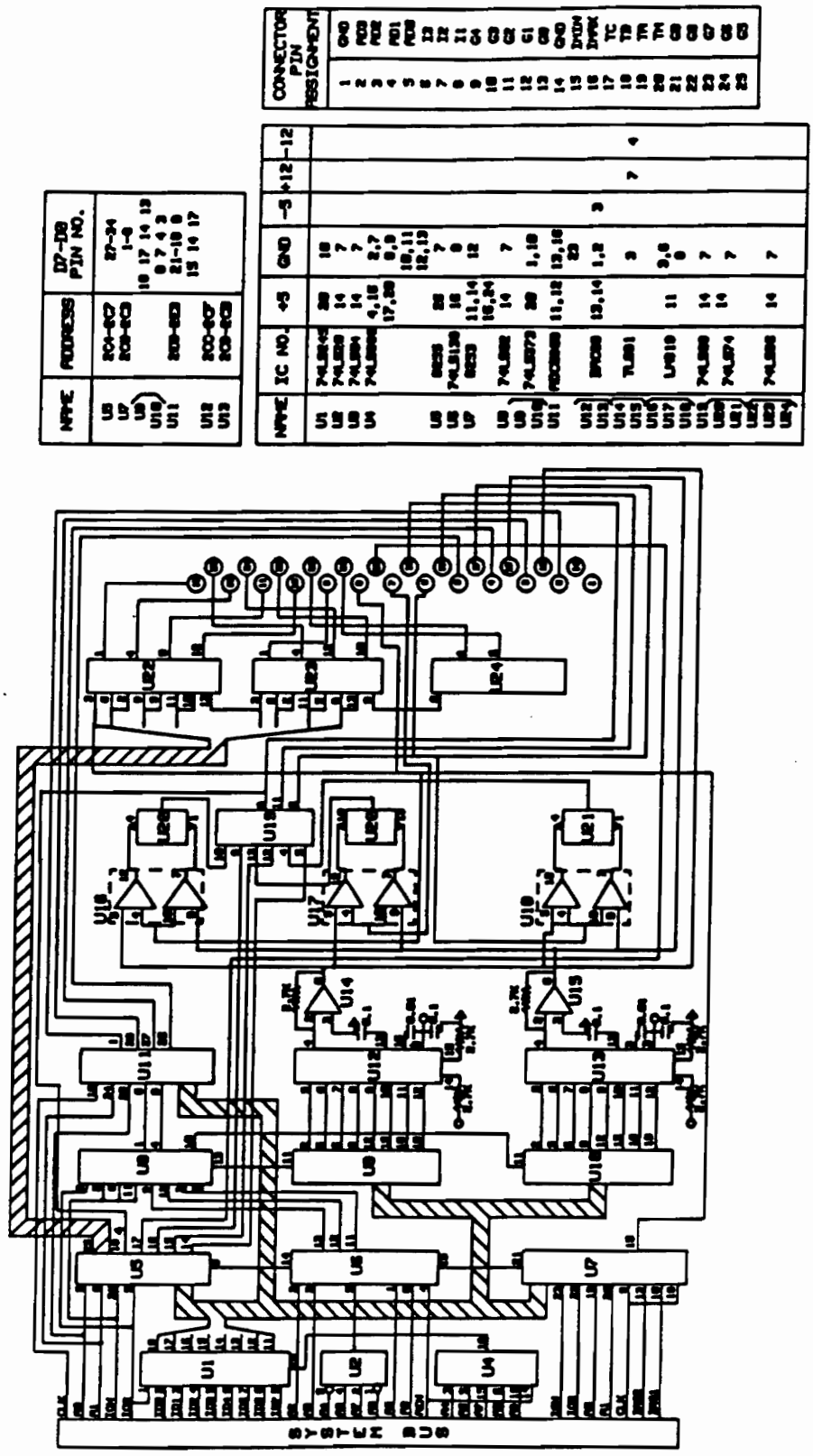
13. Mang X., Krishnan R. and Chandramouli G., *"Design and Performance of an Interactive Personal Computer Controller for Switched Reluctance Motor Drives,"* Conf. Record, IEEE IAS Annual Meeting, Pittsburgh, Oct 1987.

14. Bass J.T., Ehsani M. and Miller T.J.E., *"Simplified Electronics for Torque Control of a Sensorless Switched Reluctance Motor,"* IEEE Transactions on Industrial Electronics, Vol IE-34, No.2, pp 234-239, May 1987.

15. Goradia K.S., *"Electronic Rotor Position Sensing of Switched Reluctance Motor,"* M.S. Thesis, Department of Electrical Engineering, Virginia Polytechnic Institute & State University, Blacksburg, VA, September 1987.

16. Lindsay J.F. and Rashid M.H., *"Electromechanics and Electrical Machinery,"* Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1986.

Appendix A. Schematic of the Interface Board



Appendix A. Schematic of the Interface Board

Appendix B. Data Acquisition and Performance Evaluation Software Listings

```
/*  Data Acquisition for the Switched Reluctance Motor Drive */

#include "stdio.h"
#include "stdlib.h"
#define ARRAY 129
#define LEN 2000
#define OFF 0.10
#define VDIODE 0.7
#define RDS 0.85

typedef struct   /* define the structure type containing */
      {    /* time, position, voltage and current */
      unsigned int time; /* this structure type is used to transfer */
      int posi;          /* data from the assembler */
      int volt;
      int curr;
      } SET;

struct dataset  /* this structure is used to hold the actual*/
      {                 /* data with the proper conversion factor */
      float time;
      float posi;
      float volt;
      float curr;
      } data1[ARRAY];

extern void srm();  /* initialize the type of return */
```

```
   /* variables for the assembler program */
extern datas[];     /* the assembler structure that holds the data */

main()
{
   void save(), cls();
   srm();           /* call the controller program to run the motor */
   cls();           /* clear the screen */
   save();          /* call save() to store the actual data */
}

/* program to save the acquired data. current, voltage, position and
   time are converted to the actual value by multiplying the acquired
   value by the proper factor. */

void save()
{
   int n, m = 0;
   float volt;
   char string[6], fn[40];

   SET data[ARRAY];      /* initialize the array and pointer to */
   SET far *ptr;         /* the structure type set */
   FILE *fptr;

   printf("Enter the bus voltage: ");
   gets(string);
   volt = atof(string);
   printf("Enter the filename for storing the data: ");
   gets(fn);
   fptr = fopen(fn, "w");

   ptr = (SET *) datas;
   for (n = 0; n<ARRAY; n++)
   {
   data[n].time = ptr->time; /* Transfer all the data */
   data[n].posi = ptr->posi; /* acquired to the data */
   data[n].volt = ptr->volt; /* structure in C with the */
   data[n].curr = ptr->curr; /* multiplication factor */

   data1[n].posi = 0.35 * (float)data[n].posi; /* 90 degree = 256 */
   data1[n].time = 0.21 * (float)data[n].time; /* 1 count = 0.21 us */
   if (n > 0)
   {
   if (data1[n].posi == 0.0 && n > 100)
     data1[n].time = 0;
   else
     if (data1[n].time<data1[n-1].time) /* account for the overflow */
       {                                /* in count beyond 65536 */
       if (ptr->time < (ptr-1)->time)
```

```
        m++;
        data1[n].time += 13739.0 * m;
        }
    }
    data1[n].curr = 0.078 * (float)data[n].curr; /* 20 amperes = 256 */
    if (data1[n].curr <= OFF)
       data1[n].curr = 0;
    if (data1[n].curr > OFF && data[n].volt == 0)  /* infer the app. */
      data1[n].volt = - (volt * 1) /* phase voltage from the current */
    else                                  /* and the switch signal */
      data1[n].volt = volt * data[n].volt;
    ptr++;           /* increment pointer to point to next */
    }

    for (n=0; n<ARRAY; n++) /* store the data in the user's file */
    {
    fprintf(fptr, "%10.1f%10.2f%10.1f%10.2f\n",
          data1[n].time, data1[n].posi, data1[n].volt, data1[n].curr);
    }
    fclose(fptr);
}

void cls()
{
  int far *farptr;
  int addr;
  char ch = ' ';
  farptr = (int far *) 0xB8000000;
  for(addr = 0; addr < LEN; addr++)
    *(farptr + addr) = ch | 0x0700;
}


stack segment stack
     dw 100 dup(?)
stack ends

code segment public 'code'

DGROUP   group  _DATA
          ASSUME cs:code, ds:DGROUP, ss: stack

          PUBLIC time1

time1     PROC
          push bp          ; save registers
          push si
          push ax
          push bx
          push cx
```

```
        push dx
        mov al,0b9h
        out 21h,al
        sti
        mov  _datas[0].timer1, 0
        mov  _datas[0].volt, 0
        mov  _datas[0].curr, 0
        mov  _datas[0].posi, 80h
        mov  bl, 1
        mov  si, 0
        mov  dx, 2c3h    ; initialize card timer 2
        mov  al, 0b0h    ; for continuous counting
        out  dx, al
        mov  dx, 2c5h
chk1:   in   al, dx
        cmp  al, 80h     ; compare position with 80h to acquire
        jnz  chk1
        mov  al, 0       ; load 0 in the card timer 0
        mov  dx, 2c2h
        out  dx, al
        out  dx, al
        mov  dx, 2c5h
chk2:   in   al, dx
        and  al, 01h
        cmp  al, 01h
        jnz  chk2
        cli
        add si, 08
chk3:   in   al, dx
        mov  cl, al      ; save position in CL
        and  al, 01h
        cmp  al, 0h
        jnz  chk3
        mov  bh, cl
        mov  al, 80h
        mov  dx, 2c3h    ; latch the card timer value
        out  dx, al
        mov  dx, 2e0h    ; initiate A/D conversion at IN0
        mov  al, 0
        out  dx, al
        mov  dx, 2c4h    ; read the winding voltage
        in   al, dx
        and  al, 01h
        sti
        mov  ch, 0
        mov  ah, 0
        mov  _datas[si].volt, ax
        mov  _datas[si].posi, cx
        mov  dx, 2c2h
        in   al, dx      ; read the LSB and the MSB
```

```
            mov  cl, al
            in   al, dx
            mov  ch, al
            mov  ax, 0ffffh
            sub  ax, cx     ; subtract from ffffh to get actual value
            mov  _datas[si].timer1, ax ; save the actual value timer1
            mov  dx, 2e0h
            in   al, dx
            mov  ah, 0
            mov  _datas[si].curr, ax
            mov  dx, 2c5h
            cmp  bh, 80h
            je   rets
            inc  bl
            cmp  bl, 81h
            jne  chk2
rets:       mov al,0b1h
            out 21h,al
            pop  dx         ; restore registers
            pop  cx
            pop  bx
            pop  ax
            pop  si
            pop  bp
            ret

time1       ENDP

code        ends

_DATA       segment byte public 'DATA'
param       STRUC
timer1      DW      ?
posi        DW      ?
volt        DW      ?
curr        DW      ?
param       ENDS

_datas      param   129 DUP (<>)

            PUBLIC  _datas

_DATA       ENDS

            END


/*  Performance Evaluation of a Switched Reluctance Motor Drive */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.1416
#define ARRAY 129
#define OFF 0.10
#define STATP 6
#define ROTP 4
#define LEN 2000
#define RPHASE 3.63
#define fw_215 15.5
#define fw_390 31.1
#define fw_605 53.7
#define fw_797 75.9
#define fw_1008 102.5
#define VDIODE 0.7
#define RDS 0.85

/* define an array of structure which contains inductance as a function
   of position for many currents. The currents are approximately in
   steps of 0.25 A up to 5.0 A and 0.5 A up to 10 A. Position varies
   from  0 - 45 degrees in steps of 5 degrees */

struct indpos
   {
   float pos;
   float ind;
   };

struct profile
   {
   float amp;
   struct indpos value[10];
   } induct[31];

/* define an array of structure which contains the input data - time in
   us, position in degrees, voltage in volts and current in amperes */

typedef struct
       {
       float time;
       float posi;
       float volt;
       float curr;
       } DAT;

DAT data[ARRAY];
float torq[ARRAY], powr[ARRAY], work[ARRAY], time[ARRAY], temp[ARRAY];
float posn[ARRAY], tot_torq[ARRAY], pow_d, p_out, irms, powin, t_av;
float cu_loss, core_loss, fw_loss, tot_loss, conv_loss, rpm, eff;
```

```
float rip, fs[ARRAY][2], freq[ARRAY], fsmag[ARRAY];
char xlbl1[] = "TIME (MSEC)", ylbl1[] = "PHASE CURRENT (A)";
char ylbl2[] = "PHASE POWER (W)", ylbl3[] = "PHASE VOLTAGE (V)";
char xlbl2[] = "POSITION (DEG)", ylbl4[] = "TOTAL INST. TORQUE (Nm)";
char ylbl5[] = "INST. TORQUE/PHASE (Nm)";
char ylbl6[] = "HARMONIC MAGNITUDE (Nm)";
char xlbl3[] = "FREQUENCY (Hz)";
extern void plots(); /* declare the plotting routine as external */

main()
{

   FILE *fptr1, *fptr2;
   void rms(), power(), loss(), print_screen(), store();
   float speed();
   int m, n;
   char fn[40], in;

   if ( (fptr2 = fopen("ind.dat", "r")) == NULL)
     {  /* declare the file pointer for the inductance profile data */
     printf("\tCan't open file ind.dat containing inductance profile");
     exit(0);
     }

   for (n = 0; n < 31; n++)  /* read the inductance profile data into
                        the structure induct */
     {
     fscanf (fptr2, "%f\n", &induct[n].amp);
     for (m = 0; m < 10; m++)
       fscanf (fptr2, "%f%f\n",
              &induct[n].value[m].pos, &induct[n].value[m].ind);
     }

  do
   {
   printf("\n*************************************************\n");
   printf("    Please enter the file name for the input data      \n");
   printf("*************************************************\n\n");
   gets(fn);
   if ( (fptr1 = fopen(fn, "r")) == NULL) /* declare the file pointer*/
     {                                    /* for the logged data */
     printf("\tCan't open file %s. Program terminates.\n", fn);
     exit(0);
     }
   printf("\nComputation takes 30 seconds on an IBM AT. Please wait\n");
   for (n=0; n<ARRAY; n++) /* read the logged data into the structure */
      {
      fscanf(fptr1, "%f%f%f%f\n",
            &data[n].time, &data[n].posi, &data[n].volt, &data[n].curr);
      if (data[n].volt > 0)                 /* subtract MOSFET drop */
```

```
      data[n].volt -= 2.0 * RDS * data[n].curr; /* during on time */
     else if (data[n].volt < 0)                 /* subtract DIODE drop */
      data[n].volt -= 2.0 * VDIODE;            /* during regeneration */
     time[n] = data[n].time * 0.001;
     posn[n] = data[n].posi;
     }
  rpm = speed(); /* call speed() to compute the speed of rotation */

  torque(); /* call torque() to compute the instantaneous developed
               torque per phase */
  power(); /* call power() to compute inst. phase power input */
  rms(); /* call rms() to calculate the rms current */
  loss();/* call loss() to calculate different losses */

  do
    {
    printf("\n*************************************************\n");
    printf("\t1. Display the performance parameters          \n");
    printf("\t2. Plot the performance characteristics        \n");
    printf("\t3. Store the result                            \n");
    printf("\t4. Quit                                \n");
    printf("*************************************************\n");
    printf("Please enter your choice (Do not hit <return>)\n");
    in = getche();

    switch (in)
      {
      case '1': print_screen(); /* display performance parameters on */
                break;          /* the terminal */
      case '2': draw();  /* plot the various machine characteristics */
                break;
      case '3': store();        /* store the performance data */
                break;
      case '4': break;          /* quit performance evaluation */
      }
    }
    while (in != '4');
  fclose(fptr1);
  printf("\n*************************************************\n");
  printf("\tDo you have another set of input data (Y or N)?  \n");
  printf("*************************************************\n");
  in = getche();
  }
 while((in == 'Y') || (in == 'y')); /* repeat if there is another */
 fclose(fptr2);                                  /* data set */
}

/* this function calculates the instantaneous torque developed / phase
   and the average torque developed by the motor by taking into account
   the number of stator and rotor poles. In order to compute the torque,
```

```
   the workdone by the machine in moving from the unaligned position to
   the aligned position and back to the unaligned position is determined
   the total workdone is calculated by adding up all the workdone by the
   motor in moving from position A to position B which has been acquired
   by the data acquisition software. the workdone is calculated from the
   current vs. flux linkages characteristics drawn as a function of
   position. the currents are approximated to the nearest current value
   for which the inductance is known. the flux linkages for a particular
   position is extrapolated from the adjacent two positions. since the
   positions and thecurrents are recorded in very small steps, the error
   is negligible. the trapezoidal integration method is used to compute
   the area between two curves */

torque()
{
  int k, l, m, n, disp1, disp2;
  float psi[31][11], tmpcur1, tmpcur2, fact, actcur, actpsi, tot_work;
  float area1, area2, trap_area, psipre, psicur, lastcur, lastpsi;
  float posrad, torqsum = 0;
  void ripple(int);

  actcur = lastcur = actpsi = lastpsi = area1 = area2 = 0.0;
  for (m = 0; m < 31; m++)
    {
    psi[m][0] = induct[m].amp;
    for (n = 1; n < 11; n++)
      psi[m][n] = psi[m][0] * induct[m].value[n-1].ind;
    }

  for (n = 0; n < ARRAY; n++)
    {
    k = l = m = area1 = 0;
    if (data[n].posi > 45.0)
      data[n].posi = 90.0 - data[n].posi;
    if (data[n].curr > OFF)
      {
      while (induct[k].amp < data[n].curr)
       k++;
      if ((induct[k].amp-data[n].curr) > (data[n].curr-induct[k-1].amp))
       k--;
      while (induct[k].value[l].pos < data[n].posi)
       l++;
      fact = (data[n].posi - induct[k].value[l-1].pos) /
            (induct[k].value[l].pos - induct[k].value[l-1].pos);
      actpsi = (psi[k][l+1] - psi[k][l]) * fact + psi[k][l];
      actcur = induct[k].amp;
      psipre = 0;
      for (m = 0; m <= k; m++)
       {
       psicur = (psi[m][l+1] - psi[m][l]) * fact + psi[m][l];
```

```
    if (m == 0)
      area1 += 0.5 * induct[m].amp * psicur;
    else
      area1 += 0.5 * (induct[m].amp - induct[m-1].amp) *
             (psicur + psipre);
    psipre = psicur;
    }
   }

if ((area1 == 0 && area2 != 0) || (n > 1 && data[n-1].curr == 0))
  work[n] = 0;
else
  {
  if (actcur > lastcur)
   {
   if (lastcur != 0.0)
   trap_area = 0.5 * (actcur - lastcur) * (actpsi + lastpsi);
   work[n] = area1 - area2 - trap_area;
   }
  else
   {
   trap_area = 0.5 * (lastcur - actcur) * (actpsi + lastpsi);
   work[n] = area1 - area2 + trap_area;
   }
  }

  if (n > 0 && data[n].posi != 0.0 && data[n+1].posi != 0.0)
   {
   posrad = PI * (data[n].posi - data[n-1].posi) / 180;
   if (posrad < 0.0)
     posrad = - posrad;
   torq[n] =  work[n] / posrad;
   }
  else
   torq[n] = 0;
  tot_torq[n] = torq[n];
  area2 = area1;
  lastcur = actcur;
  lastpsi = actpsi;

}
for (n = 1; n < ARRAY; n++)
  {
  if (data[n].time != 0)
   m = n;
  }
  disp1 = m / 3;
  disp2 = 2 * m / 3;

for (n = 0; n <= m; n++)
```

```
      {
       tot_torq[n] = (n + disp1 <= m) ? (tot_torq[n] + torq[n + disp1])
                   : (tot_torq[n] + torq[n + disp1 - m]);
       tot_torq[n] = (n + disp2 <= m) ? (tot_torq[n] + torq[n + disp2])
                   : (tot_torq[n] + torq[n + disp2 - m]);
        torqsum += tot_torq[n] * (data[n].time - data[n-1].time);
      }
    t_av = torqsum / data[m].time;
    pow_d = t_av * 2.0 * PI * (rpm) / 60.0;
    ripple(m);
}

/* this function calculates the torque ripple and the fourier spectrum
   of the torque function. a discrete fourier transform is applied on
   the torque function to determine the magnitude of the average value
   and the harmonics of the fourier spectrum */

void ripple(m)
int m;
{
  int i, j;
  float temp = 0;
  for (i = 0; i < m; i++)
    {
    fs[i][0] = 0;
    fs[i][1] = 0;
    for (j = 0; j < m; j++)
      {
      fs[i][0] += torq[j] * cos ( (double) 2 * PI * i * j / m);
      fs[i][1] += torq[j] * sin ( (double) 2 * PI * i * j / m);
      }
    fs[i][0] /= m / 3.0;
    fs[i][1] /= m / 3.0;
    fsmag[i] = sqrt((double) fs[i][0] * fs[i][0] + fs[i][1] * fs[i][1]);
    if (i > 0)
      {
      fsmag[i] /= i;
      temp += fsmag[i] * fsmag[i];
      }
    freq[i] = 1000000 * i / data[m].time * 3;
    }
    temp = sqrt( (double) temp);
    rip = temp / fsmag[0] * 100;
}

/* this function calculates the speed of rotation of the motor from
   the logged data */

float speed()
{
```

```
  float rps, rpm;
  int n = 1;

  while(data[n].time != 0 && n < 129)
    n++;
  rps = 1000000 / ( data[n-1].time * 4);
  rpm = rps * 60;
  return(rpm);
}

/* this function calculates the instantaneous power / phase and hence
   the average power input of the machine */

void power()
{
  float powsum = 0;
  int n;

  powin = 0;
  for (n = 1; n < ARRAY; n++)
    {
    if (data[n].time == 0)
      break;
    else if (data[n].curr < OFF)
      powr[n] = 0;
    else
      powr[n] = data[n].volt * data[n].curr;
    powsum += powr[n] * (data[n].time - data[n-1].time);
    }
  powin = 3. * powsum / data[n-1].time;
}

/* this function computes the rms current per phase */

void rms()
{
  int n = 1;
  float isqr = 0;

  do
    {
    if (data[n].curr > OFF)
      isqr += data[n].curr * data[n].curr;
    }
  while (data[++n].time != 0);

  isqr /= --n;
  irms = sqrt( (double)isqr);
}
```

```
/* this function draws a characteristic depending upon the choice of the
   user. torque, power input, phase voltage, phase current and the
   fourier spectrum can be drawn with respect to either position or
   time */

draw()
{
   int m, n = 0;
   void cls();

   cls();
   do
     {
     printf("\n************************************************************");
     printf("\n\t1. time vs. current\n\t2. time vs. input power\n");
     printf("\t3. time vs. voltage\n");
     printf("\t4. position vs. current\n");
     printf("\t5. position vs. input power\n");
     printf("\t6. position vs. voltage\n");
     printf("\t7. position vs. torque/phase\n");
     printf("\t8. position vs. total instantaneous torque\n");
     printf("\t9. frequency spectrum of torque function\n");
     printf("\t0. quit plotting\n");
     printf("************************************************************\n");
     printf("Please enter your choice (0-9) (Do not hit <return>)\n");
     n = getche();
     switch(n)
       {
       case '1':
       case '4': for (m = 0; m < ARRAY; m++)
                   temp[m] = data[m].curr;
                   if (n == '1')
                   plots(time, temp, ARRAY, xlbl1, ylbl1);
                 else
                   plots(posn, temp, ARRAY, xlbl2, ylbl1);
                 break;
       case '2':
       case '5': if (n == '2')
                   plots(time, powr, ARRAY, xlbl1, ylbl2);
                 else
                   plots(posn, powr, ARRAY, xlbl2, ylbl2);
                 break;
       case '3':
       case '6': for (m = 0; m < ARRAY; m++)
                   temp[m] = data[m].volt;
                   if (n == '3')
                   plots(time, temp, ARRAY, xlbl1, ylbl3);
                 else
                   plots(posn, temp, ARRAY, xlbl2, ylbl3);
                 break;
```

```
        case '7': plots(posn, torq, ARRAY, xlbl2, ylbl5);
                  break;
        case '8': plots(posn, tot_torq, ARRAY, xlbl2, ylbl4);
                  break;
        case '9': plots(freq, fsmag, ARRAY, xlbl3, ylbl6);
                  break;
        case '0': break;
        default : printf("\nInvalid entry. Try again\n");
                  break;
        }
      }
    while(n != '0');
    cls();
}

/* this function calculates the copper losses, core losses, friction and
   windage losses and the converter losses. from these losses, the
   output power and efficiency are also determined */

void loss()
{
  int n = 1;
  float inst_loss, sum = 0;

  cu_loss = 3 * irms * irms * RPHASE;
  core_loss = powin - cu_loss - pow_d;
  if (rpm < 215)
    fw_loss = 0.0719 * rpm;
  else if (rpm < 390)
    fw_loss = fw_215 + 0.089 * (rpm - 215);
  else if (rpm < 605)
    fw_loss = fw_390 + 0.105 * (rpm - 390);
  else if (rpm < 797)
    fw_loss = fw_605 + 0.1156 * (rpm - 605);
  else if (rpm < 1008)
    fw_loss = fw_797 + 0.126 * (rpm - 797);
  else
    fw_loss = fw_1008 + 0.129 * (rpm - 1008);
  p_out = pow_d - fw_loss;
  do
    {
    if (data[n].volt > 0)
      inst_loss = 2.0 * RDS * data[n].curr * data[n].curr;
    else if (data[n].volt < 0)
      inst_loss = 2.0 * VDIODE * data[n].curr;
    else
      inst_loss = 0;
      sum += inst_loss * (data[n].time - data[n-1].time);
    }
  while (data[++n].time != 0);
```

```
  conv_loss = 3 * sum / data[--n].time;
  eff = 100.0 * p_out / powin;

}

/* this function prints the performance parameters on the terminal */

void print_screen()
{
  void cls();

  cls();
  printf("\n*****************************************************************\n");
  printf("\tElectromagnetic torque  = %6.2f Nm\n", t_av);
  printf("\tPower Developed \t= %6.1f W\n", pow_d);
  printf("\tTorque ripple\t\t= %6.1f %\n", rip);
  printf("\tMotor speed\t\t= %6.0f rpm\n", rpm);
  printf("\tInput Power\t\t= %6.1f W\n", powin);
  printf("\tOutput power \t\t= %6.1f W\n\tEfficiency \t\t= %6.1f %\n",
          p_out, eff);
  printf("\tRMS phase current \t= %6.2f A\n", irms);
  printf("\tCopper loss \t\t= %6.1f W\n\tCore loss \t\t= %6.1f W\n",
          cu_loss, core_loss);
  printf("\tWindage loss \t\t= %6.1f W\n", fw_loss);
  printf("\tConverter loss\t\t= %6.1f W\n", conv_loss);
  printf("*****************************************************************\n");
  printf("Hit <Return> to continue\n");
  getche();
  cls();
}

/* this function stores the performance parameters on a user specified
   file */

void store()
{
  int n, m;
  char name[40];
  FILE *fptr3;

  printf("\n*****************************************************************\n");
  printf("\tEnter the file name for storing the data
  printf(  "*****************************************************************\n");
  gets(name);
  fptr3 = fopen(name, "w");

  fprintf(fptr3,"\n*****************************************************\n");
  fprintf(fptr3, "\tElectromagnetic torque  = %6.2f Nm\n", t_av);
  fprintf(fptr3, "\tPower Developed \t= %6.1f W\n", pow_d);
  fprintf(fptr3, "\tTorque ripple\t\t= %6.1f %\n", rip);
```

```
  fprintf(fptr3, "\tMotor speed\t\t= %6.0f rpm\n", rpm);
  fprintf(fptr3, "\tInput Power\t\t= %6.1f W\n", powin);
  fprintf(fptr3, "\tOutput power \t\t= %6.1f W\n", p_out);
  fprintf(fptr3, "\tEfficiency \t\t= %6.1f %\n", eff);
  fprintf(fptr3, "\tRMS phase current \t= %6.2f A\n", irms);
  fprintf(fptr3, "\tCopper loss \t\t= %6.1f W\n", cu_loss);
  fprintf(fptr3, "\tCore loss \t\t= %6.1f W\n", core_loss);
  fprintf(fptr3, "\tWindage loss \t\t= %6.1f W\n", fw_loss);
  fprintf(fptr3, "\tConverter loss\t\t= %6.1f W\n", conv_loss);
  fprintf(fptr3,"*************************************************************\n");
  fclose (fptr3);

}

/* this function clears the terminal */

void cls()
{
  int far *farptr;
  int addr;
  char ch = ' ';
  farptr = (int far *) 0xB8000000;
  for(addr = 0; addr < LEN; addr++)
    *(farptr + addr) = ch | 0x0700;
}

void plots(xdat, ydat, n, xlbl, ylbl)
int n;
float xdat[], ydat[];
char xlbl[], ylbl[];
{
  int ystp, xstp, stepx, stepy, i, j, k, tempx, tempy, lastx,
      lasty, curx, cury, typ, frq;
  const int MINX = 47, MAXX = 559, MINY = 43, MAXY = 171, XTIC = 5;
  const int YTIC = 5;
  float xmin, xmax, ymin, ymax, span, ay, by, ax, bx, ystep, xstep, x;
  float xdata[ARRAY], ydata[ARRAY], temp;

  extern void lineqq(int, int, int, int);
  extern void locqq(int, int);
  extern void scrnqq(int);

  for (i = 0; i < ARRAY; i++)
    {
    xdata[i] = xdat[i];
    ydata[i] = ydat[i];
    }

  for (i = 0; i < ARRAY-1; i++)
    for (j = i+1; j < ARRAY; j++)
```

```
    if (xdata[i] > xdata[j])
      {
      temp = xdata[i];
      xdata[i] = xdata[j];
      xdata[j] = temp;
      temp = ydata[i];
      ydata[i] = ydata[j];
      ydata[j] = temp;
      }

ystp = (MAXY - MINY)/((YTIC - 1) * 8);
xstp = (MAXX - MINX)/((XTIC - 1) * 8);
stepy = ystp * 8;
stepx = xstp * 8;

xmin = xdata[0];
xmax = xdata[ARRAY-1];
ymax = ymin = ydata[0];
for (i = 1; i < n; i++)
  {
  ymax = (ymax >= ydata[i]) ? ymax : ydata[i];
  ymin = (ymin <= ydata[i]) ? ymin : ydata[i];
  }

ay = ((float)(MAXY - MINY)) / (ymax - ymin);
ax = ((float)(MAXX - MINX)) / (xmax - xmin);
by = -((float) (MAXY - MINY)) * ymin / (ymax - ymin);
bx = -((float) (MAXX - MINX)) * xmin / (xmax - xmin);

scrnqq(6);
lineqq(MINX, MINY, MINX, MAXY);
lineqq(MINX, MINY, MAXX, MINY);
lineqq(MINX, MAXY, MAXX, MAXY);
lineqq(MAXX, MAXY, MAXX, MINY);

ystep = (ymax - ymin) / ((float) (YTIC - 1));
tempx = MINX;
for (j = 4; j >= 0; j--)
  {
  locqq(5 + j * ystp, 0);
  temp = ymax - ystep * j;
  printf("%5.2f\n", temp);
  tempy = MINY + j * stepy;
  lineqq(tempx - 3, tempy, tempx + 3, tempy);
  }

xstep = (xmax - xmin) / ((float) (XTIC - 1));
tempy = MAXY;
for (k = 0; k < 5; k++)
  {
```

```
    locqq(22, 3 + k * xstp);
    temp = xmin + xstep * k;
    printf("%5.1f\n", temp);
    tempx = MINX + k * stepx;
    lineqq(tempx, tempy - 3, tempx, tempy + 3);
    }

    locqq(0,0);
    printf("X AXIS: %s\n", xlbl);
    locqq(1,0);
    printf("Y AXIS: %s\n", ylbl);

    lastx = MINX + xdata[0] * ax + bx;
    lasty = MAXY - ydata[0] * ay - by;
    for (j = 1; j < n; j++)
      {
      cury = MAXY - ydata[j] * ay - by;
      curx = MINX + xdata[j] * ax + bx;
      lineqq(lastx, lasty, curx, cury);
      lastx = curx;
      lasty = cury;
      }
    locqq(24, 0);
    printf("Press any key to return to main menu...");
    i = getch();
    scrnqq(3);
}
```

Vita

Mr.G. Chandramouli was born in Neyveli, India in 1965. He received his B.E. degree from PSG College of Technology, Coimbatore, India. He has been pursuing his M.S. degree in Electrical Engineering at Virginia Polytechnic Institute and State University, Blacksburg, Virginia from September 1986. His research interests are in Microprocessor-based system design and VLSI system design. He is a member of IEEE. His hobbies are tennis, shuttle badminton and traveling.